ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

"Desarrollo de aplicaciones móviles para AR.Drone Parrot en AndEngine"

Autor: Jorge Wederago Jiménez

Tutor: Oscar Ardaiz Villanueva

Pamplona, 29 de Junio de 2015

Índice

Introducción

Hoy en día la tecnología se está aplicando en diversos campos: deportes, labores de salvamento, exploración, militares, etc. Algunas de ellas, como el uso de drones, se están popularizando, lo que conlleva una reducción de costes en su producción. Estos factores permiten que nuevas funcionalidades o aplicativos aparezcan en el mercado cada día ya que, poco a poco, se van liberando distintos SDK para que la comunidad desarrolle su propio contenido para estos dispositivos. Un ejemplo sería la línea de "juguetes" dron que la empresa Parrot lleva fabricando desde 2010. También podemos encontrar otros ejemplos como el dron Lily, que sigue al portador de una pulsera a la vez que es capaz de grabar en vídeo y tomar fotos.

La novedad de este producto y la infinidad de oportunidades que nos ofrecen estos aparatos ha incentivado este proyecto.

Objeto

El objetivo principal consiste en la creación de una aplicación capaz de controlar hardware AR.Drone de Parrot. Además se busca que el control sea intuitivo, por lo que se implementará mediante el uso de sensores. A lo largo de este documento se describirá el proceso que se ha seguido para llegar a las soluciones propuestas.

El desarrollo de la aplicación se hará en el entorno Eclipse con el SDK de Android.

El estudio del protocolo de comunicación se hará mediante la aplicación Wireshark.

El interfaz gráfico se implementará con el motor AndEngine.

Se cuenta con una Tablet Nexus 7 y un AR.Drone Parrot 2.0 para realizar las pruebas.

AR.Drone Parrot

¿Qué es el AR.Drone Parrot?

El AR.Drone de Parrot es un cuadróptero comercial con una de las mejores relaciones calidad-precio del mercado. Parrot es una empresa especializada en la creación de dispositivos de manos libres que, en 2010, comenzó una línea de productos dron

Características del AR.Drone 2.0

- Registrador de vuelo GPS
- Cámara HD a 720p a 30 FPS
- Peso total de 380g con la carcasa de exteriores y 420 con la de interiores.
- Motores "inrunner" sin escobillas 14,5W 28.500 RPM
- Procesador de 1 GHz y 32 bit ARM Cortex A8 con vídeo DSP TMS320DMC64x de 800 MHz
- Linux 2.6.32
- RAM DDR2 de 1GB a 200MHz
- USB 2.0 de alta velocidad para extensiones
- Wi-Fi b g n
- Giroscopio de 3 ejes con una precisión de 2000°/seg.
- Acelerómetro de 3 ejes con una precisión de +/- 50mg
- Magnetómetro de 3 ejes con una precisión de 6°
- Sensor de presión con una precisión de +/- 10 Pa
- Sensores de ultrasonido para medir la altitud de avance
- Cámara vertical QVGA de 60 FPS para medir la velocidad de avance

Cuenta con una página web para desarrolladores en la que podemos encontrar distintas versiones del API para sus productos.

Aplicaciones controladoras de AR.Drone 2.0

Existen muchas aplicaciones controladoras para este tipo de dron. Muchas de ellas las encontramos en el propio SDK que se obtiene de la página oficial de desarrolladores.

- AR.FreeFlight: Aplicación oficial de IOS y Android.

Se trata de la aplicación más completa que podemos encontrar para estas plataformas. Dispone de una biblioteca donde almacenar todos los vídeos y fotos que tomemos con el dron así como de una plataforma digital donde los usuarios intercambian experiencias de vuelo y material multimedia.

Al pulsar "Take Off" pasamos a la pantalla de control.

Desde ella podemos intercambiar la imagen entre las cámaras del dron, grabar vídeos y sacar fotos. Además tenemos el botón de despegue (Take off) para comenzar el vuelo del dron. Durante el vuelo este botón se convierte en aterrizar (Landing). Los joysticks virtuales nos permiten controlar el dron. Por último, en la pantalla contamos con diversa información sobre el estado del dron como puede ser la batería aproximada y el estado de la red.

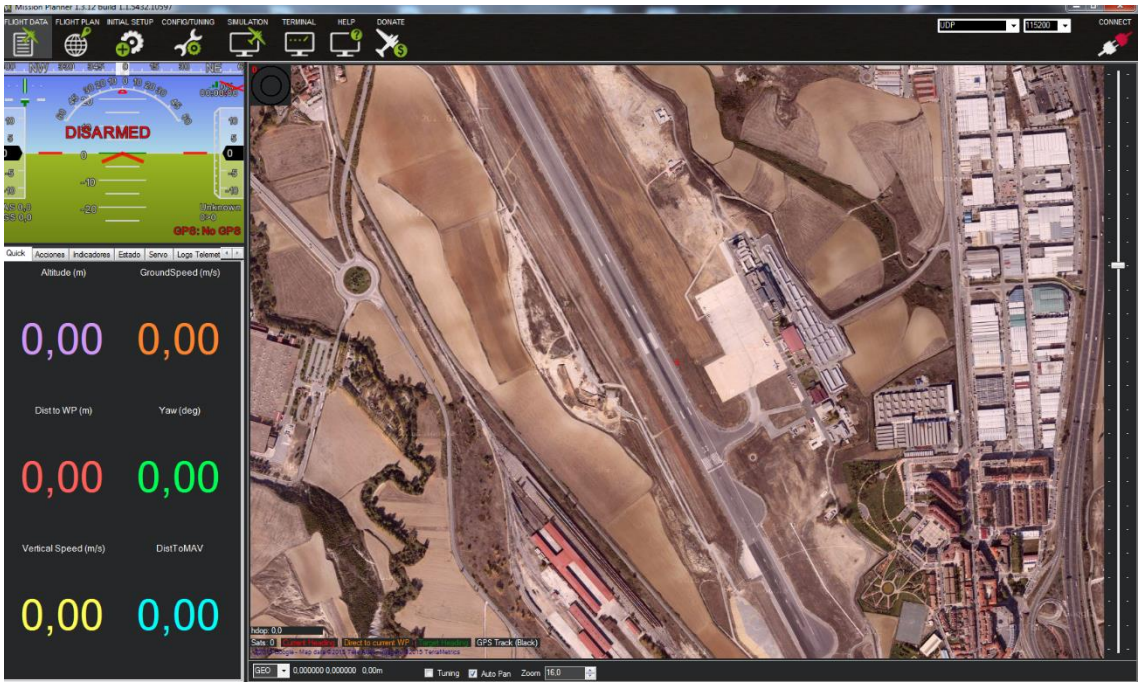
Al pulsar la tecla de configuración podemos acceder a las siguientes opciones.

Estas opciones nos permiten tener el control total y mantener el dron dentro de una serie de parámetros para evitar posibles accidentes. La más destacable de todas es poder limitar la altura máxima a la que se permite volar al dron.

Actualmente la aplicación está disponible en Windows 8.

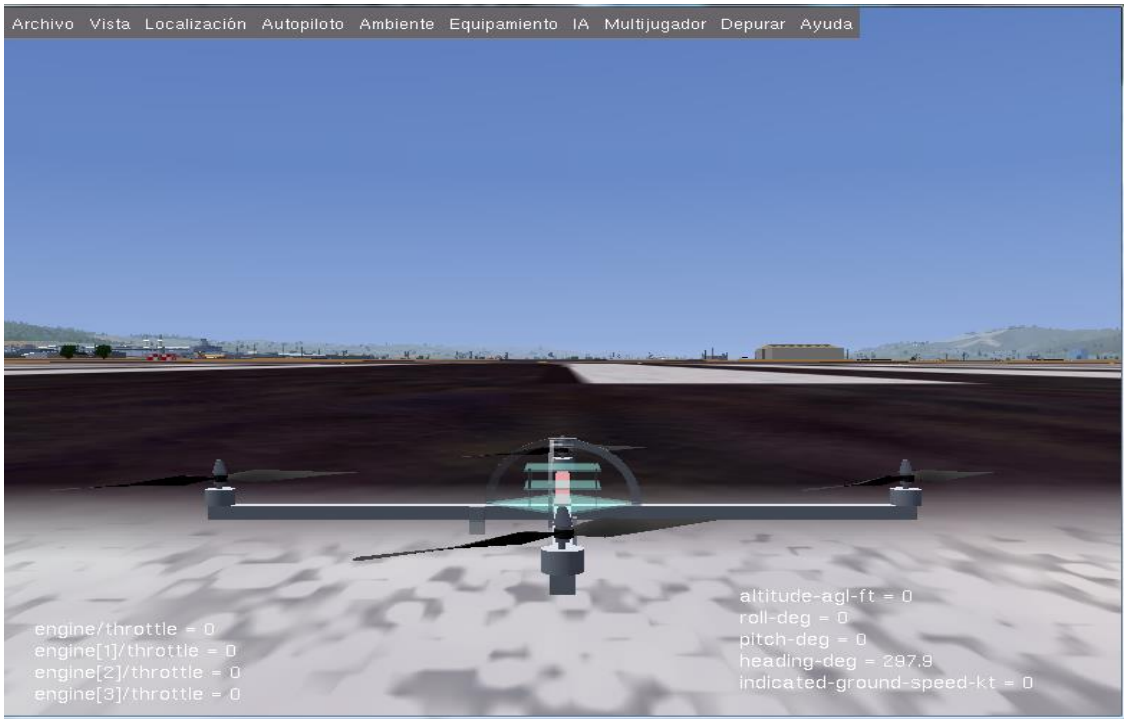
- Mission Planner y FreeFlight: Se trata de un planificador y un simulador de vuelo con los que se puede trazar planes de vuelos y simularlos en entornos 3D. Además posibilitan personalizar el protocolo de envío de información para hacer simulaciones con controladores de dron reales.

  El interfaz de Mission Planner es el siguiente:



En él podemos ver el aeropuerto de Noáin.

Por otro lado tenemos el interfaz de FlightGear, el simulador de vuelo, en el que podemos cargar el modelo 3D del objeto que queremos que realice la simulación o recorrido enviado desde Mission Planner.



AndEngine

¿Qué es AndEngine?

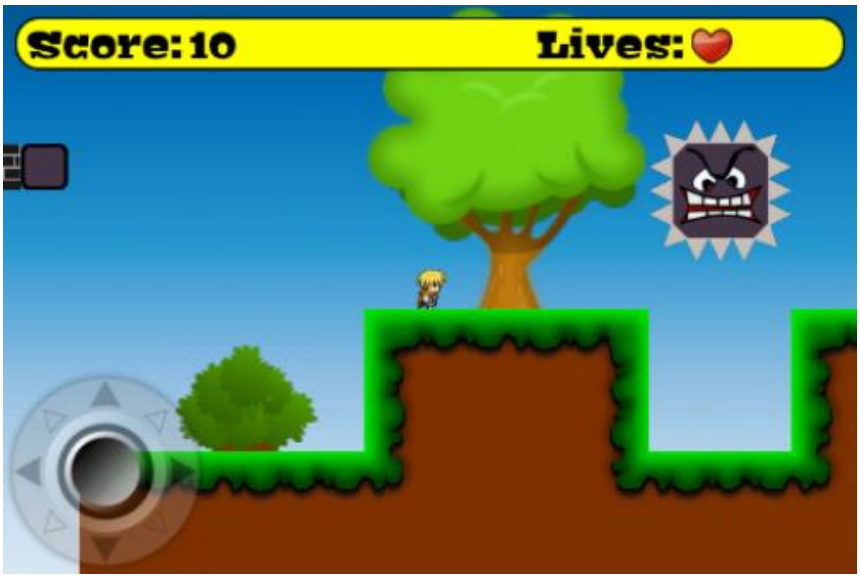
AndEngine es un motor gráfico para Android gratuito escrito en Java desarrollado por Nicolas Gramlich. Hace uso de OpenGL ES. Cuenta con todo lo necesario para la realización de juegos en 2D para dispositivos Android.

A pesar de no contar con documentación en sí, existen ejemplos desarrollados por el creador que introducen los conceptos de este motor gráfico. Como consecuencia, la mayor cantidad de información en lo relativo a problemas y soluciones se encuentra en los foros oficiales.

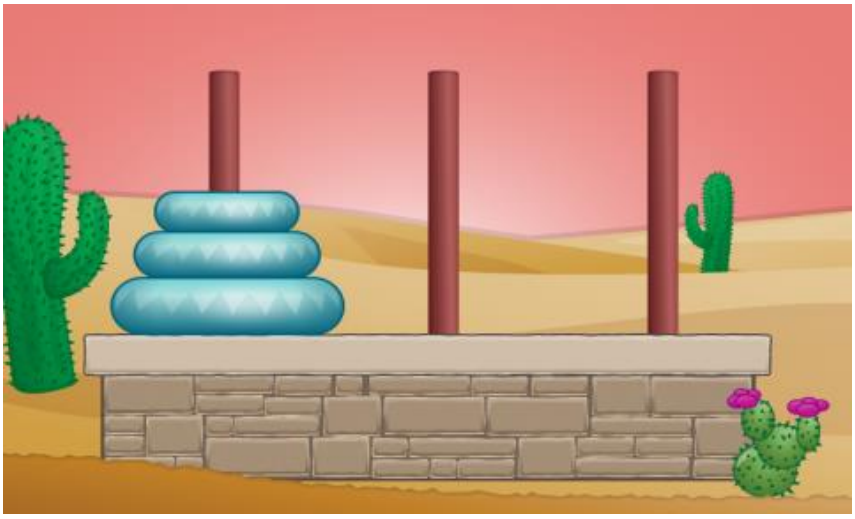
Una de sus grandes ventajas es que escala de manera automática todos los componentes de una aplicación creada en este motor. Esto lo consigue gracias a una cámara definida de modo que, sin importar el tamaño del dispositivo, todos los componentes gráficos mantienen las relaciones.

Algunos ejemplos de aplicaciones creadas con AndEngine:

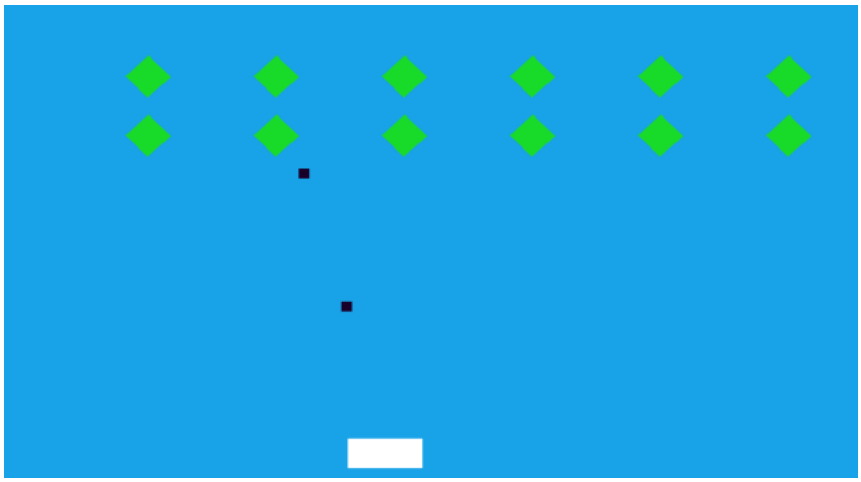
Juego de Plataformas:



Torres de Hanói:



Space Invaders:



Desarrollo de la aplicación

Conociendo el protocolo

Una vez conocidas las herramientas que se van a utilizar nos topamos con que el código que viene en el API es la aplicación oficial. Para su instalación se precisa del NDK de Android, el cual permite la carga y utilización de librerías en otros lenguajes. Esto supone un inconveniente ya que en caso de necesitar cambiar algo de estas librerías pueden surgir problemas de compilación.

A la vista de lo anterior procedemos al estudio del protocolo de comunicación entre la aplicación oficial y el dron para intentar simularlo en Android. Para ello realizamos un vuelo de prueba y guardamos los datos que se mueven en la red con la aplicación Wireshark. Se usará la aplicación "Shark" en un móvil rooteado.

De esta lectura de datos y comparándola con la información de la guía del desarrollador obtenemos la siguiente información:

Todos los comandos comienza por AT* y, en función de qué acompañe a ste comando, se tratará de un tipo de orden u otra. En un mismo paquete podemos encontrar varias instrucciones siempre y cuando quepan. Por otro lado el paquete nunca deberá exceder los 1024 caracteres o será desechado. Se transmite a través de UDP.

Todos los paquetes tienen un número de secuencia que comienza en 1 desde que se establece la conexión y si no se envía nada durante 2 segundos éste se vuelve a resetear a 1. El Dron nunca ejecutará nada que tenga un número de secuencia menor que lo recibido anteriormente salvo si es 1. Este número de secuencia lo encontramos tras el tipo de una instrucción AT*____=Nsecuencia.

Tipos de comandos:

AT*REF: Se trata del comando encargado de decidir el despegue o aterrizaje del Dron (también del de emergencia).

Si el Dron permanece en tierra los paquetes son del siguiente modo:

```
.......H 6.AT*CTR
L=22,5,0 .AT*PCMD
_MAG=23, 0,0,0,0,
0,0,0.AT *REF=24,
29071769 6.
```

Si el Dron está volando de este otro:

```
.......Q v.AT*CTR
L=12212, 5,0.AT*P
CMD_MAG= 12213,0,
0,0,0,0, 0,0.AT*R
EF=12214 ,2907182
08.
```

AT*PCMD/AT*PCMD_MAG: Los paquetes que contienen esta orden son los que gestionan el vuelo del Dron y más concretamente los encargados del control del mismo. Estos mensajes están compuestos de manera similar a los anteriores y están en TODOS los paquetes que se envían al Dron. Después del número de secuencia cuenta con otros 7 valores separados por una coma. A continuación se enumera el significado de cada uno de estos números.

1. Número de secuencia, se localiza tras el "=" que encontramos después de la orden.
2. Bandera o Flag para indicar si se están usando comandos progresivos, es decir, se permite ejecutar otras órdenes a la vez que ésta.
3. Valor entre -1 y 1 que representa inclinación lateral Izquierda-Derecha.
4. Valor entre -1 y 1 que representa inclinación Adelante-Atrás.
5. Valor entre -1 y 1 que representa la velocidad vertical.
6. Valor entre -1 y 1 que representa la velocidad angular.
7. Valor de "magneto psi". Exclusivo de la versión de la orden AT*PCMD_MAG.
8. Valor de la exactitud del "magneto psi". Exclusivo de la versión de la orden AT*PCMD_MAG.

Algunos ejemplos:

```
.......Q V.AT*CTR
L=12212, 5,0.AT*P
CMD_MAG= 12213,0,
0,0,0,0, 0,0.AT*R
EF=12214 ,2907182
08.
```

Dron estabilizado

```
.......Q ..AT*PCM
D_MAG=13 827,1,10
08164608 ,1025005
728,0,0, 0,0.AT*R
EF=13828 ,2907182
08.
```

Dron rotando

```
.......H ..AT*PCM
D_MAG=13 661,0,0,
0,104877 2972,0,0
,0.AT*RE F=13662,
29071820 8.
```

Dron en ascenso

Además de toda esta información propia de los paquetes también observamos los puertos que establecen comunicaciones.

- 5551: Conexión FTP
- Descarga un archivo de información del dron que le indica la versión del mismo así como otros datos. Como por ejemplo la altura máxima permitida al dron, …
- 5554: Puerto de NavData
    - Encargado de recibir la información enviada por el dron en cuanto a su estado.
- 5555: Puerto de Video
    - Encargado de recibir el streaming de video.
- 5556: Puerto de Comandos

    - Encargado de recibir los distintos comandos desde la aplicación
- 5559: Puerto de Control

Creación de una aplicación servidor que simule el dron

Mission Planner y FlightGear

Una primera posibilidad que se planteó fue utilizar Mission Planner y FlightGear para hacer simulaciones de vuelo.

Mission Planner iba a ser el encargado de generar las rutas, es decir, de planear el recorrido.

Con el recorrido creado el simulador gratuito de código abierto FlightGear se encargaría de simular la misión planeada.

Esta idea tuvo que ser desechada ya que a pesar de la posibilidad de unir estos dos programas mediante la creación de unos sencillos ficheros .bat resultó imposible implementar el protocolo de comunicación estudiado. La causa fue que en cualquier caso era necesario tener una placa de Arduino/Raspberry en un puerto COM que sería el encargado de simular el dron.

Implementación en Android

Finalmente se optó por hacer una aplicación servidor que simulase el dron en Android con los datos obtenidos del estudio del protocolo. Esta decisión se tomó debido a la necesidad de utilizar un dispositivo capaz de generar una red Wi-Fi a la que conectarse y, en el caso de un dispositivo Android, es una característica disponible por defecto en la mayoría de su hardware.

Clase MainActivity

Se trata de una clase sencilla que hace instancias de HiloServidor encargado de crear conexiones UDP en los puertos ya nombrados para preparar la conexión.

```
//Servidor FTP
HiloServidor server5551 = new HiloServidor(5551,this);
server5551.start();
//Servidor NAVDATA_PORT 5554
HiloServidor server5554 = new HiloServidor(5554,this);
server5554.start();
//Servidor VIDEO_PORT 5555
HiloServidor server5555 = new HiloServidor(5555,this);
server5555.start();
//Servidor CMD_PORT 5556
HiloServidor server5556 = new HiloServidor(5556,this);
server5556.start();
//Servidor CONTROL_PORT 5559
HiloServidor server5559 = new HiloServidor(5559,this);
server5559.start();
```

Clase HiloServidor

Esta clase monta un servidor UDP en el puerto pasado por parámetro. Dentro de esta misma clase diferenciamos en qué puerto se ha montado y tratamos los datos en función de cuál se trata.

Por ahora sólo distinguimos el puerto 5556 que es a donde llegan las instrucciones propias del dron. Haciendo uso de expresiones regulares filtramos los mensajes y mostramos por pantalla el tipo de comando que se nos ha enviado, así como su significado.

```
if(port==5556){
        String frase = new String(recibirDatos,0,recibirPaquete.getLength());
    Pattern p = Pattern.compile(".*AT\\*CONFIG.*");
    Matcher m = p.matcher(frase);
    if (m.find()){
        System.out.println("Mensaje de Configuración recibido "+frase);
    }
    p = Pattern.compile(".*AT\\*REF.*");
    m = p.matcher(frase);
    if (m.find()){
        System.out.println("Mensaje de estado recibido "+frase);
    }
    p = Pattern.compile(".*AT\\*TRIM.*|.*AT\\*FTRIM.*");
    m = p.matcher(frase);
    if (m.find()){
        System.out.println("Mensaje de decision de horizontalidad "+frase);
    }
    p = Pattern.compile(".*AT\\*LED.*");
    m = p.matcher(frase);
    if (m.find()){
        System.out.println("Mensaje de configuración de los LED "+frase);
    }
    p = Pattern.compile(".*AT\\*PCMD.*");
    m = p.matcher(frase);
    if (m.find()){
        System.out.println("Mensaje de movimiento del DRON "+frase);
        String[] parts = frase.split(",");
```

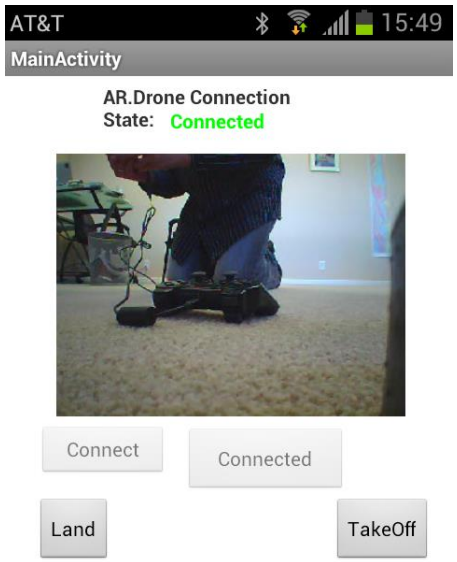
Al tratar de probar la aplicación oficial nos encontramos con que no tenemos manera de seleccionar la IP de conexión, por lo que nos resulta imposible hacer que se

conecten entre ellos. Esto, sumado a lo mencionado anteriormente sobre el NDK, puede suponer un problema. Se decide buscar una librería implementada en otro lenguaje.

Tras barajar varias posibilidades se optó por hacer uso de la implementación JavaDrone desarrollada por Codeminders.

Esta implementación buscaba convertir todo el código del API original en código java para permitir la creación de aplicaciones en este lenguaje.

Aplicación ControlTower-Android:



Como podemos observar la aplicación tan solo cuenta con 4 botones, ya que el control sobre el dron se lleva a cabo haciendo uso de un mando de Play Station 3.

Creación de la aplicación controladora

La implementación de Codeminders puede obtenerse desde un repositorio GitHub y está construida sobre Maven.

Maven es una herramienta de software para la gestión y construcción de proyectos Java que utiliza un "Project Object Model (POM)" para describir las dependencias del proyecto.

Una aplicación construida sobre Maven tiene la ventaja de que todas las referencias a librerías externas pueden ser descargadas de manera automática y, además, nos permite modificarlas al no tratarse de librerías estáticas privadas.

Maven se añade de forma similar a cualquier otro módulo que se instale en eclipse. Haremos uso del siguiente enlace:

http://rgladwell.github.com/m2e-android/updates/

Con la aplicación ya descargada se puede observar que se han creado cinco proyectos.

Se descarga el proyecto ControlTower-Android en un dispositivo Android para probarlo con el dron y vemos que somos capaces de despegar y aterrizar. Sin embargo, al observar el logcat podemos ver que tanto el decodificador de video como el de NavData fallan, por lo que la aplicación sólo manda información al dron sin recibir datos de estado ni de vídeo. Esto a priori no supone ningún problema por lo que se decide comenzar a trabajar con esta implementación.

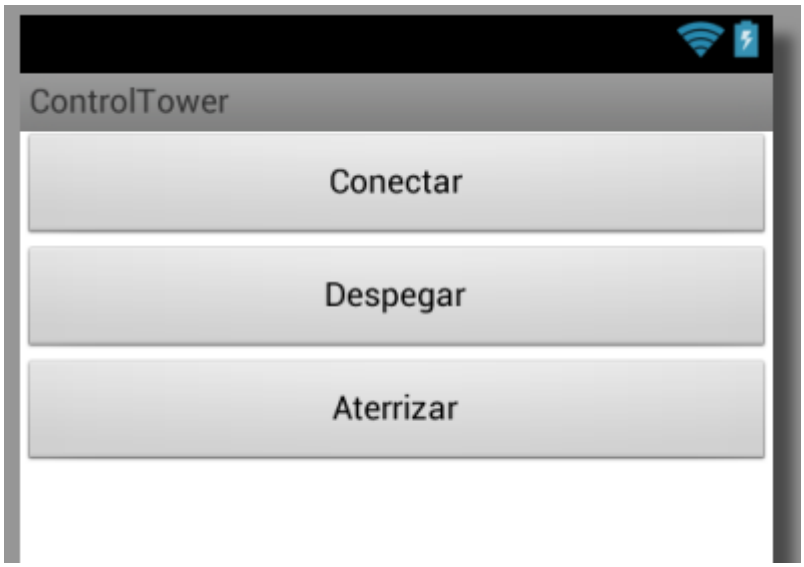
Como primer paso se crea una clase propia con interfaz que nos permita las funciones básicas de conectar, despegar y aterrizar. Nos limitamos a éstas ya que la implementación ControlTower de JavaDrone trabaja con un mando Dual Shock 3 como controlador total para el dron.

Para ello se crea una clase llamada Principal dentro del paquete ardrone de la aplicación que simula el funcionamiento del MainActivity propio de la misma. Cabe destacar DroneStarter, que es la encargada de crear el objeto dron así como de realizar la conexión a éste.

```
private class DroneStarter extends AsyncTask<ARDrone, Integer, Boolean> {

    @Override
    protected Boolean doInBackground(ARDrone... drones) {
        ARDrone drone = drones[0];
        try {
            drone = new ARDrone(InetAddress.getByAddress(ARDrone.DEFAULT_DRONE_IP), 10000, 60000);
            MainActivity.drone = drone;
            drone.connect();
            drone.clearEmergencySignal();
            drone.trim();
            drone.waitForReady(CONNECTION_TIMEOUT);
            drone.playLED(1, 10, 4);
            drone.selectVideoChannel(ARDrone.VideoChannel.HORIZONTAL_ONLY);
            drone.setCombinedYawMode(true);
            return true;
        } catch (Exception e) {
            Log.e(TAG, "Failed to connect to drone", e);
            try {
                drone.clearEmergencySignal();
                drone.clearImageListeners();
                drone.clearNavDataListeners();
                drone.clearStatusChangeListeners();
                drone.disconnect();
            } catch (Exception ex) {
                Log.e(TAG, "Failed to clear drone state", ex);
            }

        }
        return false;
    }

    protected void onPostExecute(Boolean success) {
        if (success.booleanValue()) {
            droneOnConnected();
        } else {
            state.setTextColor(Color.RED);
            state.setText("Error");
            connectButton.setEnabled(true);
        }
    }
}
```

Esta clase se puede importar a nuestro proyecto directamente (con los cambios necesarios en las referencias). Añadimos los métodos encargados de llamar a las funciones de conectar, despegar y aterrizar.

Asignamos un layout sencillo a la clase que hemos creado.



Y ya tenemos nuestra clase lista para hacer pruebas.

Al hacer una prueba de conexión y despegue obtenemos los siguientes resultados en el logcat.

```
System.out              Mensaje de Configuración recibido AT*CONFIG=1,"general:video_enable
                        ","TRUE"
System.out              Mensaje de Configuración recibido AT*CONFIG=2,"video:bitrate_contro
                        l_mode","1"
System.out              Mensaje de estado recibido AT*REF=3,290717952
System.out              Mensaje de decision de horizontalidad AT*FTRIM=4
System.out              Mensaje de configuración de los LED AT*LED=5,1,1084227584,4
System.out              Mensaje de Configuración recibido AT*CONFIG=6,"video:video_channel"
                        ,"0"
System.out              Mensaje de estado recibido AT*REF=7,290717952
System.out              Mensaje de decision de horizontalidad AT*FTRIM=8
```

Como podemos observar los datos enviados por la clase DroneStarter son recibidos por la aplicación servidor que hemos creado. Es más, los paquetes están formateados como se había visto en el capítulo sobre el estudio del protocolo.

Por lo tanto ya tenemos un servidor que es capaz de simular el funcionamiento del dron y, por otro lado, comprobamos el funcionamiento de la aplicación que hemos obtenido gracias al código de Codeminders.

Cabe subrayar que para que la conexión con el servidor se realice hubo que cambiar una serie de datos en la clase ARDrone.java de Codeminders.

- Modificar el parámetro DEFAULT_DRONE_IP a la IP del dispositivo al que nos queremos conectar. En este caso a la IP 192.168.43.1 ya que usamos una Tablet Nexus 7 como servidor.
- Y en el método connect() de este mismo archivo añadimos

```
changeState(State.CONNECTING);
changeState(State.DEMO);
```

para garantizar la conexión ya que el cambio en el estado no se produce si no hay un dron real. Esta modificación se lleva a cabo al recibir un dato que se obtiene desde el dron mediante NavData.

No se trata de una solución óptima porque pase lo que pase el cliente siempre se "conectará" haya algo o no al otro lado. No se ha encontrado ninguna solución mejor para el problema y no supone un gran inconveniente si se es consciente de ello.

Estas medidas se llevaron a cabo ya que en un paso previo a la primera conexión entre cliente y servidor, se intentó simular el envío de paquetes de NavData porque era imposible establecer la conexión entre ellos ya que se quedaba en el estado "conectando". Esta simulación se realizó desde la clase servidor en el puerto 5554.

Como primera prueba, ya que resultaba imposible saber a qué se referían todos y cada uno de los bits que conforman el paquete, se utilizó uno obtenido durante el estudio de protocolo y se usó literalmente como contestación cada vez que se recibiese algún tipo de información en dicho puerto.

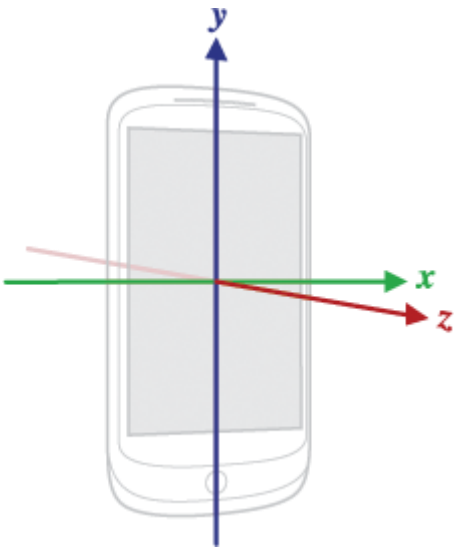
```
if( port==5554){
    recibirDatos = recibirPaquete.getData();
    if (recibirPaquete.getLength()==4){

        byte[] udpMsg= {(byte)0x88,(byte)0x77,(byte)0x66,(byte)0x55,(byte)0x

        DatagramSocket ds = null;
        ds = new DatagramSocket();
        DatagramPacket dp;
        InetAddress serverAddr = InetAddress.getByName("192.168.43.189");
        dp = new DatagramPacket(udpMsg, udpMsg.length, serverAddr, 5554);
        ds.send(dp);
    }
}
```

Sin embargo, incluso con estos cambios, resultó imposible realizar la conexión. Tras estudiar el funcionamiento de la clase AR.Drone y NavDataDecoder se pudo determinar que no se daba la condición necesaria para cambiar el estado a DEMO que es el requisito para establecer la conexión, por lo que se decidió cambiarlos manualmente para facilitarla.

Ya tenemos la funcionalidad básica, el siguiente paso consiste en implementar el control de vuelo.

Implementando control de vuelo

Para la implementación de vuelo se opta por usar movimientos naturales para controlar el dron en vez de usar un control analógico. Para ello se hará uso de los sensores que todo dispositivo Android tiene, más concretamente se usará el acelerómetro. En la siguiente imagen podemos observar los ejes considerados en estos dispositivos.



El objetivo es que una inclinación del dispositivo hacia el eje Y positivo se convierta en movimiento lateral hacia la izquierda por parte del dron y el inverso sea hacia la derecha. Un movimiento positivo en el eje X sea avanzar y uno negativo retroceder.

Para hacer uso de estos datos es necesario implementar SensorEventListener en la clase, de este modo los métodos onSensorChanged y onAccuracyChanged estarán disponibles. Los valores de los sensores se obtienen del SensoreEvent de onSensorChanged de modo que si accedemos a ellos tenemos las lecturas de los 3 ejes. Del mismo modo podemos obtener un timestamp que nos indica el momento en el que se ha leído el valor.

Al hacer una simple prueba que saque las lecturas del sensor por pantalla vemos que, a pesar de estar apoyado sobre una mesa, el valor de los sensores varía y no es 0. Por lo tanto debemos tratar los datos para que nos sean útiles. Una primera solución consiste en poner una variable que denominamos inicialización de tipo booleana. Esta variable inicializada a falso pasará a verdadero en la primera ejecución de onSensorChanged de modo que tenemos un valor inicial. Este valor lo utilizaremos para hacer un cálculo que nos permita determinar si el movimiento ha sido lo suficientemente significativo como para tenerlo en cuenta, es decir, vamos a filtrar los valores según un valor “ruido” que vamos a elegir. Calculamos la diferencia entre el último valor de la variable y el actual y mediante un valor absoluto obtenemos el valor

de diferencia. Si este valor es mayor al "ruido" que hemos definido lo tendremos en cuenta, si no pondremos el valor a 0 para ignorarlo.

En el código se ha tomado el valor de NOISE como 2 ya que se considera que es un valor lo suficientemente alto como para filtrar bien el ruido.

```
if (!mInitialized) {
    mLastX = x;
    mLastY = y;
    mLastZ = z;

    mInitialized = true;
} else {

    deltaX = Math.abs(mLastX - x);
    deltaY = Math.abs(mLastY - y);
    deltaZ = Math.abs(mLastZ - z);

    aux1=mLastX - x;
    aux2=mLastY - y;
    aux3=mLastZ - z;

    if (deltaX < NOISE)
        deltaX = (float)0.0;
    if (deltaY < NOISE)
        deltaY = (float)0.0;
    if (deltaZ < NOISE)
        deltaZ = (float)0.0;}
```

Una vez realizada esta operación nos basta con comparar los valores entre los 3 ejes para determinar qué tipo de movimiento se ha dado. Por lo tanto la comparación entre los distintos delta nos dará el resultado. Dado que nos interesa comprobar movimientos diagonales también comprobaremos estos casos.

Una vez sabemos el tipo de movimiento que se ha dado procedemos a mandar la instrucción de movimiento al dron. Este comando ya ha sido comentado en el estudio del protocolo y en la librería de Codeminders viene determinado como *drone.move(left_right_tilt, front_back_tilt, vertical_speed, angular_speed)*

- left_right_tilt = valor de inclinación lateral
- front_back_tilt = valor de inclinación frontal
- vertical_speed = valor de ascenso/descenso
- angular_speed = valor de rotación

Los valores para estos campos van desde -1 a 1 indicando con dicho valor la dirección del movimiento. Esta información la podemos encontrar en la guía del desarrollador y en la clase ControllerThread.java de la librería de Codeminders. En ella vemos cómo toman los valores de los joysticks de Play Station y los dividen entre 128. Dado que estos valores van desde -128 a 128 se obtiene el grado de inclinación de la palanca y el correspondiente valor que podríamos enviar.

Por simplificación nuestro programa va a mandar valores 1 y -1 para los movimientos.

Todo lo comentado lo implementamos de la siguiente manera.

```
if (deltaX>0 && deltaY>0){
    if (aux1>0 && aux2>0){
        drone.move(1,1,0,0);
    }else if(aux1<0 && aux2>0){
        drone.move(-1,1,0,0);
    }else if(aux1>0 && aux2<0){
        drone.move(1,-1,0,0);
    }else{
        drone.move(-1,-1,0,0);
    }
}else if (deltaX>0 && deltaY==0){
    if (aux1<0){
        drone.move(-1,0,0,0);
    }else{
        drone.move(1,0,0,0);
    }
}else if (deltaX==0 && deltaY>0){
    if (aux2<0){
        drone.move(0,-1,0,0);
    }else{
        drone.move(0,1,0,0);
    }
}
```

De este código cabe mencionar que si nos fijamos bien en las instrucciones, los movimientos responden a la lectura de los sensores con el dispositivo en landscape. Además, como se puede observar, no se trata el eje Z a pesar de tener los valores guardados. Dado que vamos a trabajar con un interfaz en 2D, que es lo que AndEngine nos proporciona, no tenemos estos valores en cuenta. Por otro lado no se encontró ningún movimiento o gesto natural para este eje.

Otras ideas que surgieron durante la implementación

Obtener las distancias recorridas según la aceleración

A la par que se implementó esta solución, se estudió la posibilidad de dibujar en un interfaz el recorrido que realizaba el dron suponiendo que la aceleración que el acelerómetro indicaba se aplicase al movimiento del dron. Para ello se usó el código para el cálculo de aceleración lineal que encontramos en Google Developers.

http://developer.android.com/reference/android/hardware/SensorEvent.html

```
public void onSensorChanged(SensorEvent event){
  // In this example, alpha is calculated as t / (t + dT),
  // where t is the low-pass filter's time-constant and
  // dT is the event delivery rate.

  final float alpha = 0.8;

  // Isolate the force of gravity with the low-pass filter.
  gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
  gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
  gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

  // Remove the gravity contribution with the high-pass filter.
  linear_acceleration[0] = event.values[0] - gravity[0];
  linear_acceleration[1] = event.values[1] - gravity[1];
  linear_acceleration[2] = event.values[2] - gravity[2];
}
```

Una implementación similar a ésta y el uso de timestamps nos permiten calcular la velocidad y el espacio recorrido mediante un sencillo cálculo diferencial.

```
//Controlamos el timestamp
long ahora = event.timestamp;
long tiempo = ahora-tAnt;
tAnt=ahora;
//ponemos el tiempo en segundos
float temporal=(float) (tiempo*0.000000001);
//calculamos la velocidad actual
velocidad[0]=vAnt[0]+linear_acceleration[0]*temporal;
velocidad[1]=vAnt[1]+linear_acceleration[1]*temporal;
velocidad[2]=vAnt[2]+linear_acceleration[2]*temporal;
//aplicamos la formula para obtener los valores de x,y,z para ello tenemos en cuenta lo que se han desplazado
//no solo por aceleración si no también por velocidad filtramos para valores de aceleracion superiores a 1
if (Math.abs(linear_acceleration[0])>1 && Math.abs(linear_acceleration[1])>1){//movimientos diagonales
    if (linear_acceleration[0]>0 && linear_acceleration[1]>0){//Diagonal x+   y+
        System.out.println("Diagonal x+   y+");
    }else if (linear_acceleration[0]<0 && linear_acceleration[1]>0){//Diagonal x-   y+
        System.out.println("Diagonal x-   y+");
    }else if(linear_acceleration[0]>0 && linear_acceleration[1]<0){//Diagonal x+   y-
        System.out.println("Diagonal x+   y-");
    }else{//Diagonal x-   y-
        System.out.println("Diagonal x-   y-");
    }
}else if (Math.abs(linear_acceleration[0])>1 && Math.abs(linear_acceleration[1])<1){//solo consideramos en x
    punto[0]=(float) (pAnt[0]+0.5*linear_acceleration[0]*(temporal*temporal)) + velocidad[0]*temporal;
    if (linear_acceleration[0]>0){
        System.out.println("Movimiento x positivo");
    }else if (linear_acceleration[0]<0){
        System.out.println("Movimiento x negativo");
    }
}else if(Math.abs(linear_acceleration[0])<1 && Math.abs(linear_acceleration[1])>1) {//solo consideramos en y
    punto[1]=(float) (pAnt[1]+0.5*linear_acceleration[1]*(temporal*temporal)) + velocidad[1]*temporal;
    if (linear_acceleration[1]>0){
    }else if (linear_acceleration[1]<0){
    }
}
//Actualizamos el punto anterior
pAnt[0]=(float) punto[0];
pAnt[1]=(float) punto[1];
pAnt[2]=(float) punto[2];
//Actualizamos la velocidad anterior
vAnt[0]=velocidad[0];
vAnt[1]=velocidad[1];
vAnt[2]=velocidad[2];
System.out.println("X= "+punto[0]+" Y= " +punto[1]+" Z= "+ punto[2]);
```

El resultado de este experimento no era realista, ya que no tenemos manera de mandar al dron el espacio concreto que debe recorrer, por lo tanto las realidades no coincidían.

Implementar la posibilidad de grabar y retransmitir órdenes

Una vez se pudo mandar instrucciones al dron surgió la idea de grabar un recorrido para su posterior envío. Para ello se implementó un botón nuevo en el interfaz que cambiaba el valor de un flag que denominamos "grabación". "Grabación" activa la opción que comienza a grabar los comandos en vez de enviarlos. Se hizo uso de dos clases nuevas. Por un lado la clase Orden.java en la cual almacenamos las instrucciones ya que representa una estructura.

```
public Orden (int x,int y,int z,long timestamp) {
    this.x=x;
    this.y=y;
    this.z=z;
    this.timestamp=timestamp;
}
```

Esta clase cuenta con sus correspondientes métodos get().

Por otro lado tenemos Imprimir.java. Se trata de un hilo que es el encargado de mandar al dron las instrucciones en los mismos intervalos que se grabaron. Esto es posible gracias a que en las órdenes también guardan un timestamp.

Desarrollando el interfaz gráfico con AndEngine

Con esta nueva funcionalidad ya desarrollada y lista para usar se decide comenzar a crear el interfaz gráfico que implemente todas las funcionalidades planteadas. Para ello se hará uso del motor gráfico AndEngine.

Dado que el interfaz se va a desarrollar en este motor debemos tener en cuenta que no se utilizan layouts por lo que todos los interfaces planteados hasta el momento no son válidos.

Definiendo un programa que usa AndEngine

Así como en Android se extiende de la clase Activity, en AndEngine se extiende de otras clases. Para estos interfaces vamos a extender de la clase SimpleBaseGameActivity que es la actividad más básica de las que tenemos en esta librería.

```
public class Interface extends SimpleBaseGameActivity{

    @Override
    public EngineOptions onCreateEngineOptions() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected void onCreateResources() {
        // TODO Auto-generated method stub

    }

    @Override
    protected Scene onCreateScene() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Al extender de esta clase se nos generarán 3 métodos.

EngineOptions

En este método se especifican las opciones del motor gráfico a definir. Aquí se declara la cámara a usar, es decir el tamaño de la pantalla que se va utilizar para el interfaz. Como parámetro de salida tiene EngineOptions que serán las opciones del motor gráfico definido.

```
public EngineOptions onCreateEngineOptions() {

    final Camera camera = new Camera(0, 0, CAMERA_WIDTH, CAMERA_HEIGHT);

    return new EngineOptions(true, ScreenOrientation.LANDSCAPE_FIXED, new RatioResolutionPolicy(CAMERA_WIDTH, CAMERA_HEIGHT), camera);
}
```

Para los interfaces a desarrollar se va a tomar un tamaño de cámara de 800x480 ya que vamos a trabajar en LandScape (horizontal).

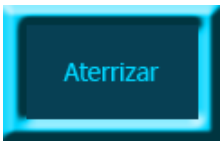
onCreateResources

Este método será el encargado de cargar los recursos a utilizar en nuestro proyecto. Todo lo que vaya a ser usado en el mismo debe ser declarado aquí ya que AndEngine hace una precarga de estos en la memoria virtual del dispositivo. Existen muchas maneras de definir estos recursos, pero en este caso, se ha optado por crear un Atlas en el que cargaremos todas las imágenes que vamos a necesitar.

El uso de un Atlas se debe a que en esta clase de motores gráficos, las imágenes, a partir de ahora texturas, se cargan en espacios con tamaños múltiplos de 2. Por esta razón es muy fácil desperdiciar espacio de memoria si no se generan con un tamaño concreto. Por ejemplo, si se tiene una textura de 260x260 se debería definir en memoria como una textura de 512x512 ocupando casi el doble de espacio de lo que realmente necesita. Este problema se soluciona haciendo uso de un Atlas que nos permite declarar

texturas de gran tamaño e ir cargando dentro de ella texturas de menor tamaño para maximizar el uso del espacio.

```
botones = new BuildableBitmapTextureAtlas(this.getTextureManager(),4096,128);

bAterrizar= BitmapTextureAtlasTextureRegionFactory.createFromAsset(botones, this, "aterrizar.png");
```



El código anterior crea un atlas llamado “botones” de tamaño 4096x128 que servirá de base para cargar todas las texturas necesarias para el proyecto. Además se carga la textura bAterrizar en él usando la imagen “aterrizar.png”. La justificación del tamaño de 4096x128 se debe a que se cargan 30 texturas de tamaño 120x75.

Por último cargamos el atlas.

```
botones.build(new BlackPawnTextureAtlasBuilder<IBitmapTextureAtlasSource, BitmapTextureAtlas>(0, 0, 0));
botones.load();
```

onCreateScene

Ya se ha definido el tamaño de la pantalla y los recursos, ahora sólo falta definir la pantalla del interfaz y éste es el método encargado de ello.

Primero se define el encargado de actualizar la pantalla. Por norma general se utiliza un FPSLogger.

```
this.mEngine.registerUpdateHandler(new FPSLogger());

final Scene scene = new Scene();
scene.setBackground(new Background(0.09804f, 0.6274f, 0.8784f));
```

Además del FPSLogger se ha declarado la escena que es el parámetro de salida y se ha añadido un color de fondo a la pantalla. Estos valores para el color de fondo son valores entre 0 y 1, indican el grado de saturación en RGB.

Dado que se quiere implementar las funcionalidades ya mencionadas a lo largo del proyecto mediante botones, hay que definirlos.

Existen al menos 8 constructores de botones en AndEngine. Para este proyecto se utilizará el constructor que nos permite poner los siguientes parámetros.

- Posición X → Posición en X siendo el punto (0,0) la esquina superior izquierda.
- Posición Y → Similar a X, pero en Y.
- Textura normal →Textura del botón sin pulsar.
- Textura pulsada →Textura del botón sin pulsar.
- Textura desactivada → Textura del botón en estado deshabilitado.
- VertexBufferedObjectManager → Controlador de los objetos en el interfaz.

Se ha optado por el uso de este constructor ya que permite definir los 3 estados posibles para un botón.

```
conectar = new ButtonSprite(0, 202.5f, bConectar, bConectarPulsado,bConectarDesactivado, this.getVertexBufferObjectManager());
scene.attachChild(conectar);
```

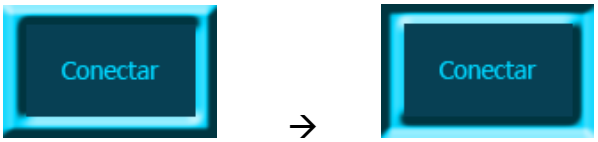
Cada vez que se declara un objeto para la escena, debe añadirse a ésta mediante attachChild().

A nivel visual, el botón ya forma parte del interfaz, sin embargo, aún no tiene ningún tipo de funcionalidad. Para dotarle de esta funcionalidad debemos declarar un onTouchListener para el objeto ButtonSprite. Esto se podría hacer usando otro constructor, pero se ha optado por hacerlo manualmente.

El resultado sería el siguiente:

```
conectar = new ButtonSprite(0, 202.5f, bConectar, bConectarPulsado,bConectarDesactivado, this.getVertexBufferObjectManager()){
    public boolean onAreaTouched(TouchEvent pTouchEvent, float pTouchAreaLocalX, float pTouchAreaLocalY){
        if(pTouchEvent.isActionDown()){
        }
        else if(pTouchEvent.isActionUp()){
        }
    return super.onAreaTouched(pTouchEvent, pTouchAreaLocalX, pTouchAreaLocalY);
    }
};
scene.attachChild(conectar);
scene.registerTouchArea(conectar);
```

Como se puede observar hay que registrar también el evento a la escena. Gracias al constructor utilizado en la creación del botón, la animación de pulsación es automática.



Ahora tan solo resta añadir la llamada a las distintas funciones o métodos bien al pulsar o bien al soltar el botón. Se considera que es mejor llamar a los métodos al liberar la pulsación.

```
conectar = new ButtonSprite(0, 0, bConectar, bConectarPulsado,bConectarDesactivado, this.getVertexBufferObjectManager()){
    public boolean onAreaTouched(TouchEvent pTouchEvent, float pTouchAreaLocalX, float pTouchAreaLocalY){
        if(pTouchEvent.isActionDown()){
        }
        else if(pTouchEvent.isActionUp()){
            inicializar();
        }
    return super.onAreaTouched(pTouchEvent, pTouchAreaLocalX, pTouchAreaLocalY);
    }
};
scene.attachChild(conectar);
scene.registerTouchArea(conectar);
```

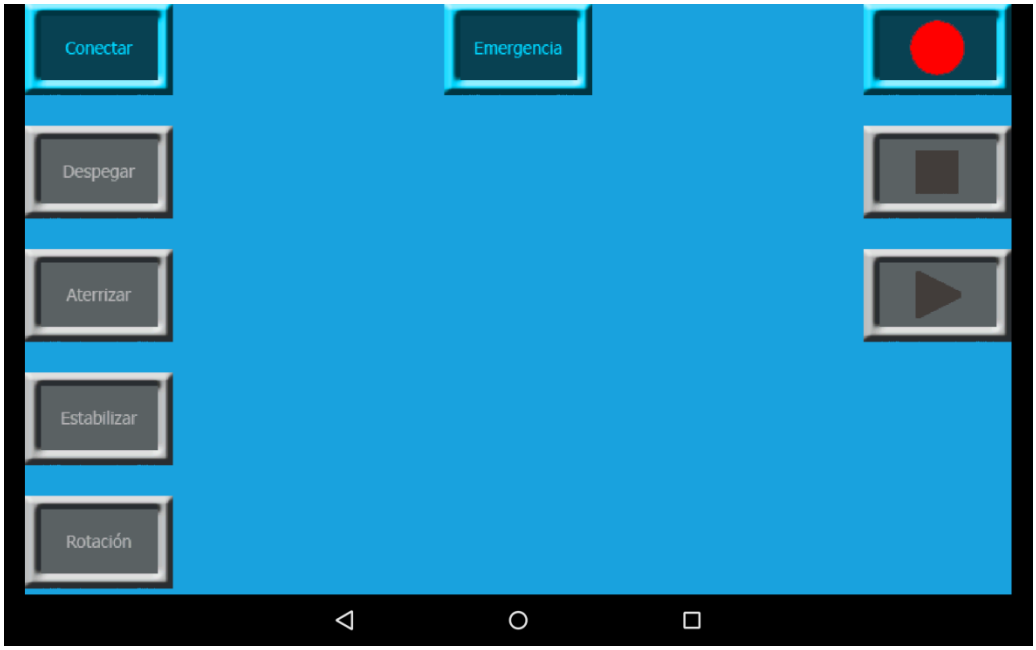
Del mismo modo que se ha desarrollado la creación e implementación del botón conectar se implementan todos los demás botones.

Posibles interfaces

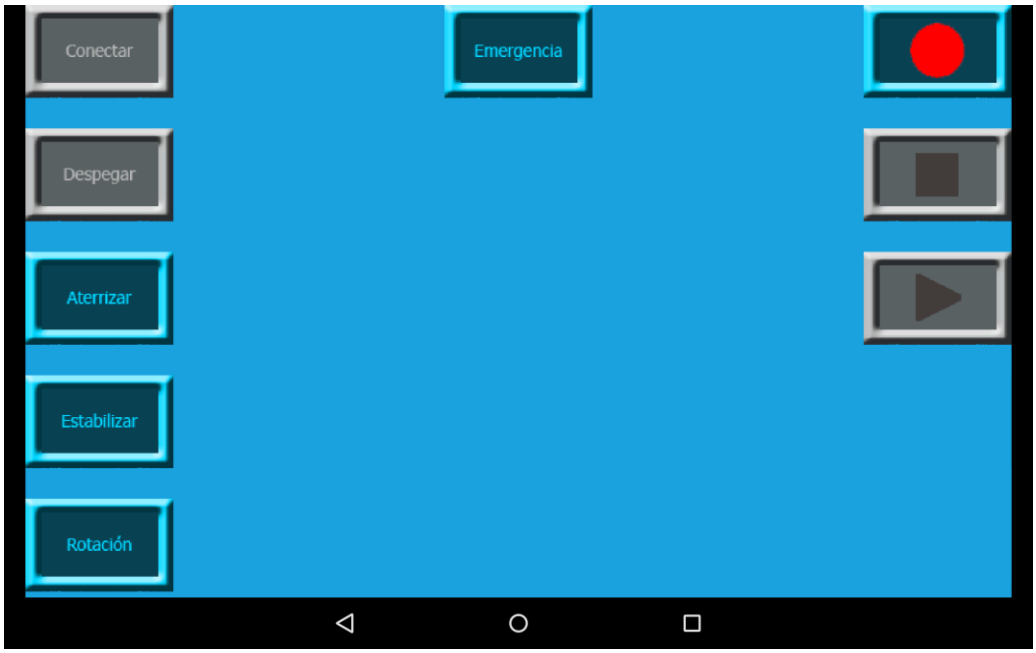
Interfaz controlador

Dado que el objetivo es hacer un interfaz sencillo e intuitivo en el que destaque la funcionalidad más que el aspecto visual nos encontramos ante dos posibilidades.

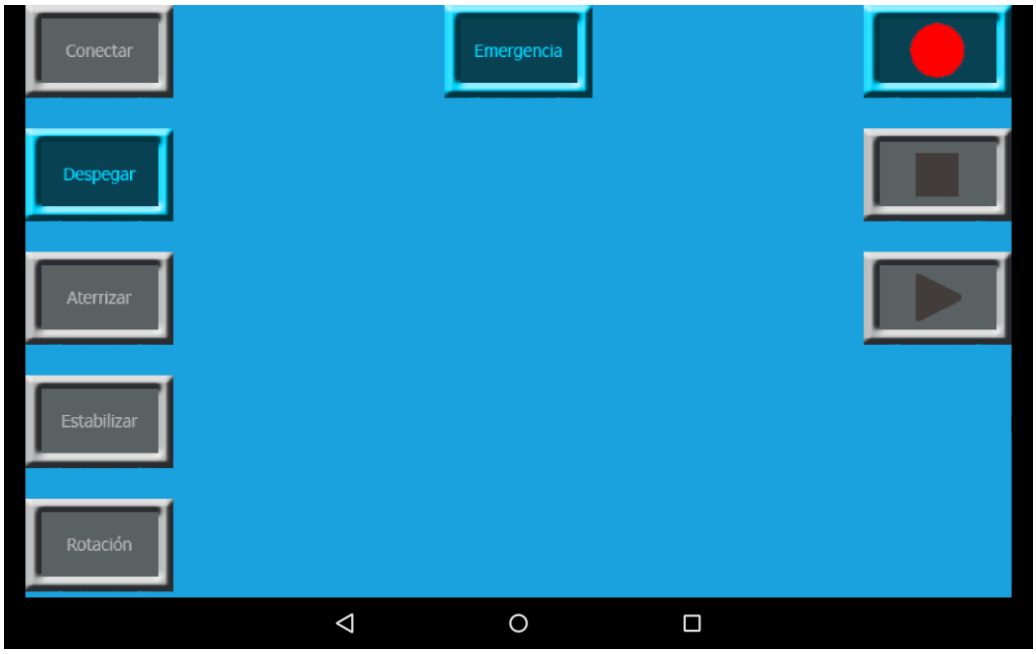
1. Un interfaz completo con todas las opciones a la vista.



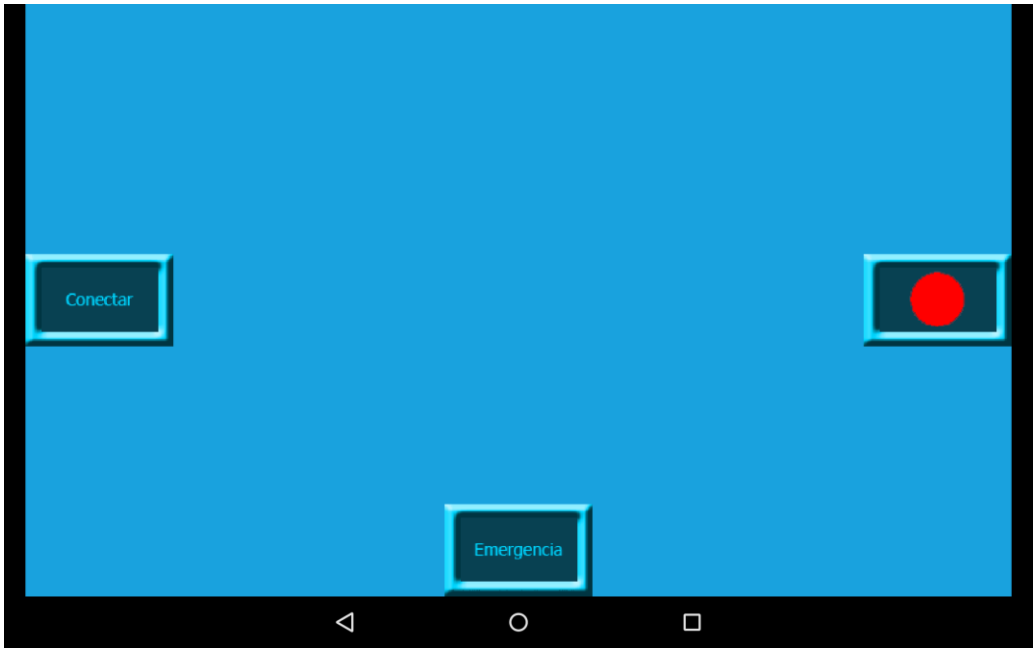
Que en función de lo que se vaya pulsando active y desactive las opciones correspondientes.



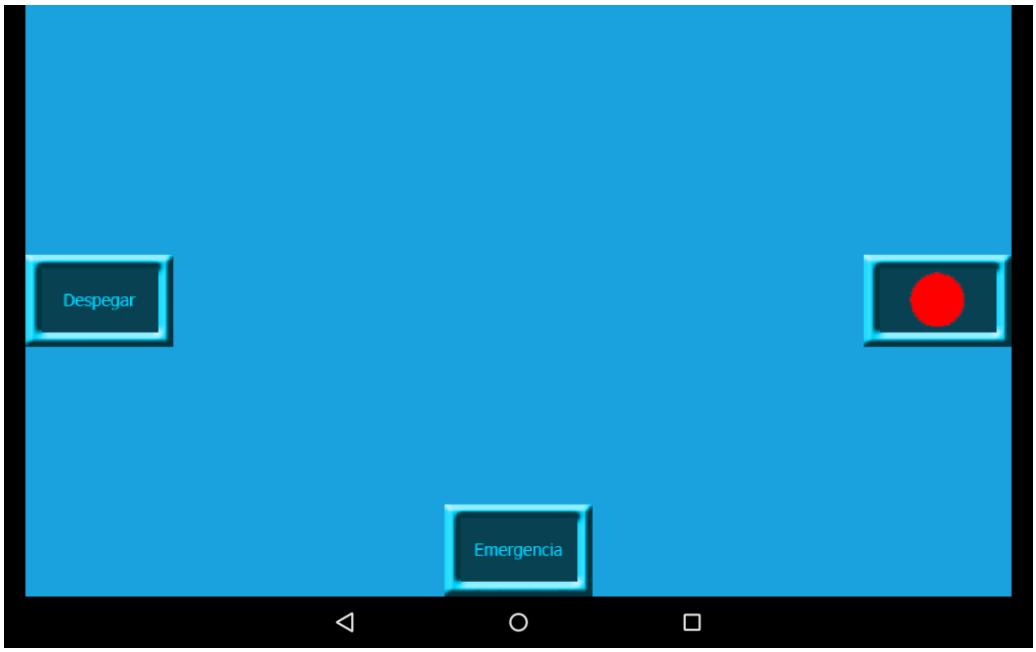
De modo que a pesar de tener todo a la vista, sabemos en todo momento qué acciones se pueden realizar o no.



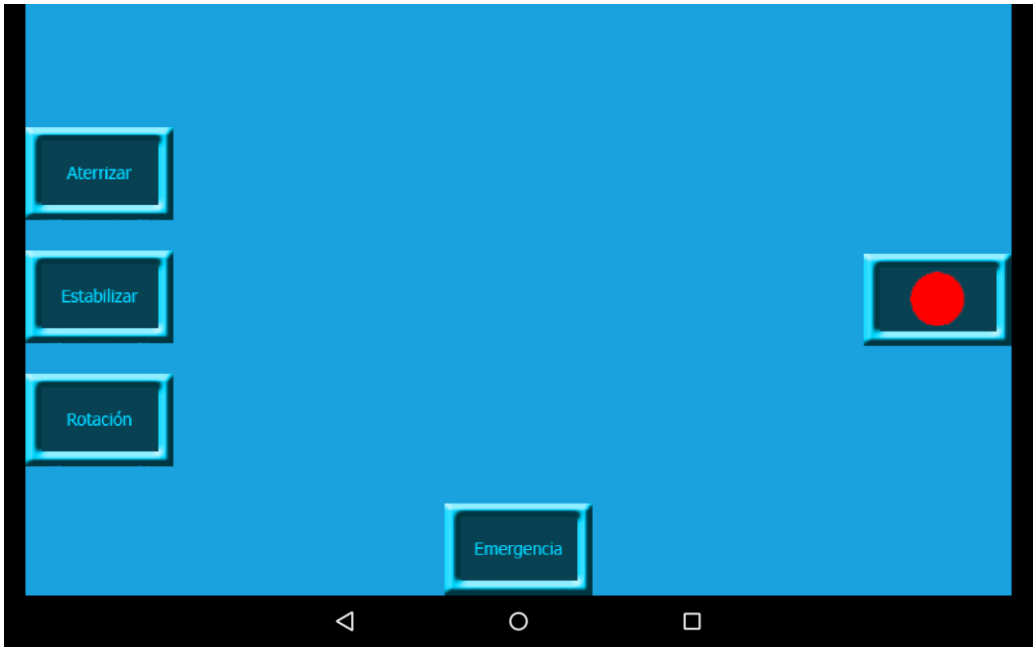
2. O un interfaz que tan solo muestre lo que es posible en cada momento.



Al pulsar conectar nos aparece la opción de despegar.



Tras despegar aparecen las opciones de aterrizar, estabilizar o rotar.



Interfaz simulador

Este interfaz se definió en un estilo similar al anterior. Además se le añadió la funcionalidad de dibujar el recorrido que se indicaba al dron desde el controlador. Para dibujar el recorrido se creó una función llamada "hacerMovimiento" que crea rectángulos en las coordenadas proporcionadas a la llamada. Para ver la progresión del recorrido se va cambiando la coloración de los rectángulos. Estas coordenadas consisten en un incremento de 10 en el eje X y/o Y en función de lo recibido desde la aplicación controladora.

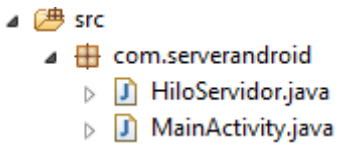
Además se ha añadido un botón que limpia la pantalla para tener mayor claridad de los recorridos.

Aplicaciones finales

Aplicación Servidor – ServEngine

En este proyecto se combina todo lo desarrollado hasta ahora.

Se compone de dos clases.*



MainActivity

Al igual que la original, ya comentada, es la encargada de crear los hilos servidores para escuchar lo que se recibe desde el dron. Sin embargo en esta versión también implementa todo lo necesario para dibujar por pantalla lo que se recibe y un botón para reiniciar el interfaz.

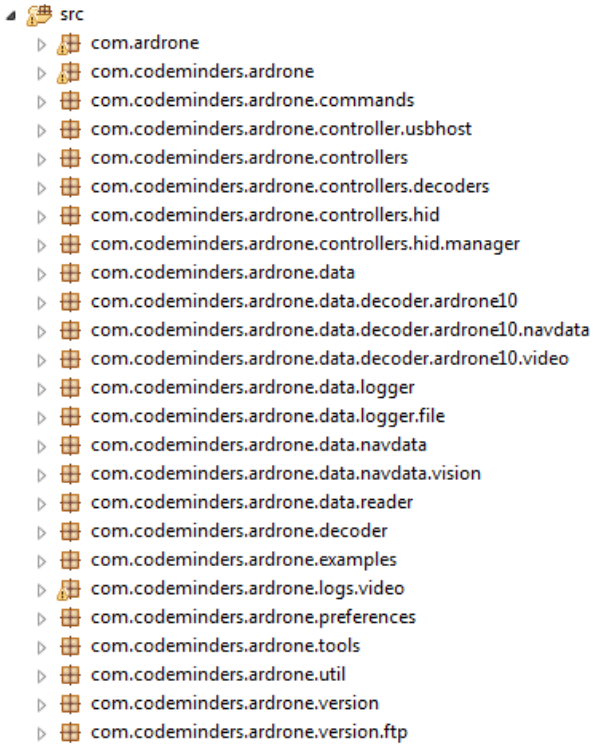
HiloServidor

Al igual que MainActivity se trata de la misma clase que se creó anteriormente, pero se le ha dotado de la capacidad de discernir de qué tipo de mensaje de movimiento se trata y de llamar a la clase que dibuja en MainActivity para tener una representación gráfica de lo que se está recibiendo.

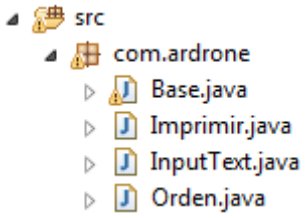
Aplicación Controladora - ArDrone

Antes de comenzar a trabajar en este proyecto se optó por usar todos los paquetes que se utilizaban en los distintos módulos del proyecto original Maven para evitar problemas de dependencias. Se tomó esta decisión debido a que resultó imposible añadir al proyecto original de Codeminders AndEngine. Además, al intentar tomar sólo lo necesario para el proyecto se generaba una serie de errores de dependencias entre clases que no se podían solucionar de otra manera. Así, en la carpeta src tenemos lo siguiente:

**El orden de aparición de las clases en las imágenes se debe a que Eclipse ordena las clases por orden alfabético. Las clases se explican en función de cómo están relacionadas.*



Todas las clases desarrolladas para el proyecto se encuentran en com.ardrone.*



Orden

Es la misma clase que se desarrolló para el proceso de grabación y reproducción de instrucciones en diferido.

Imprimir

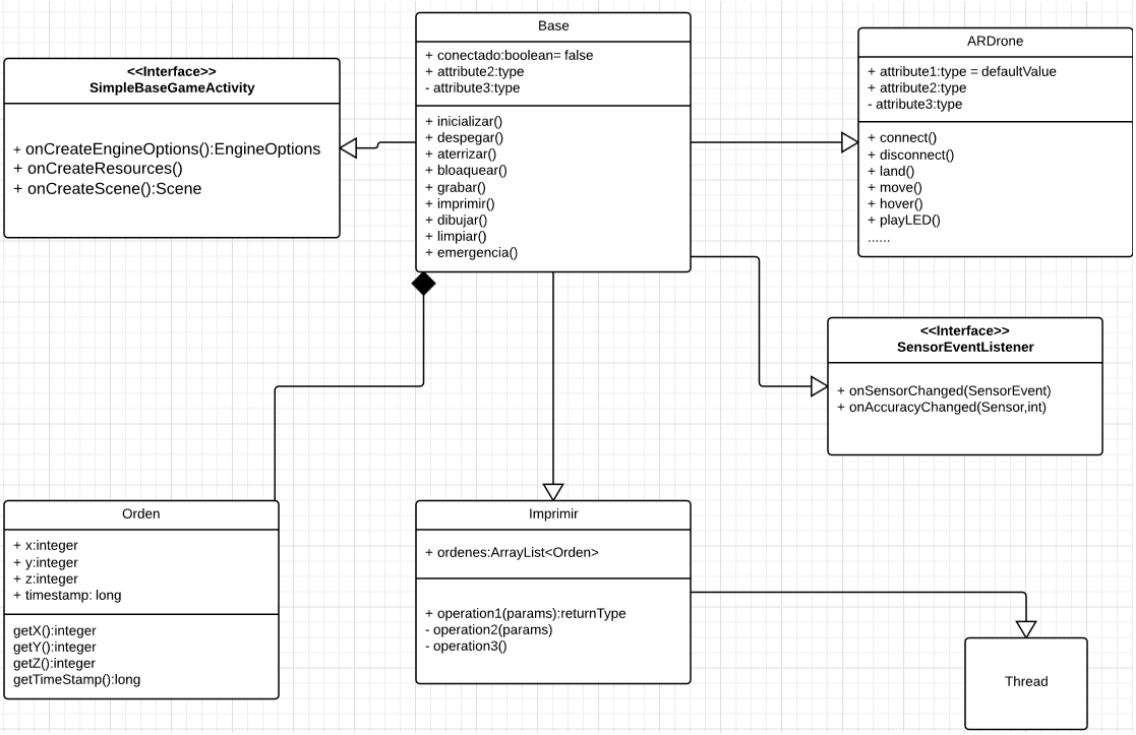
La clase original, pero en este caso se le permite modificar el estado de los botones del interfaz principal “Base.java” para bloquear los botones durante el envío.

InputText

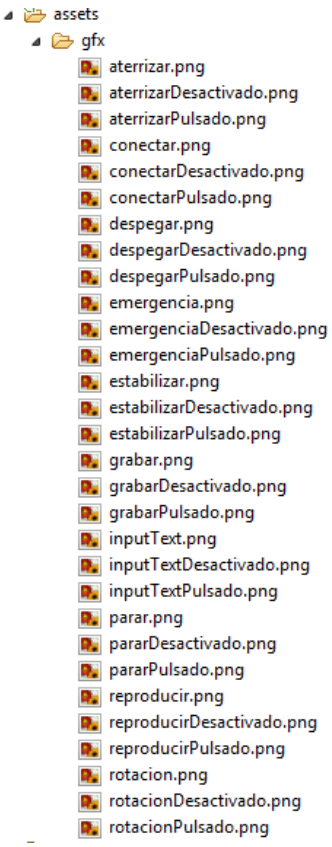
Se trata de una clase que modifica un ButtonSprite de AndEngine y permite un campo de introducción de texto. Esto es algo de lo que el motor carece y hay que implementarlo manualmente. En este caso se utiliza el “dialog” propio de Android para tener esta posibilidad.

Base

Clase principal que contiene los cálculos para el control del dron, contiene el interfaz principal e implementa todos los métodos utilizados en el mismo.



Todas las imágenes necesarias para el proyecto se han añadido a la carpeta assets/gfx.

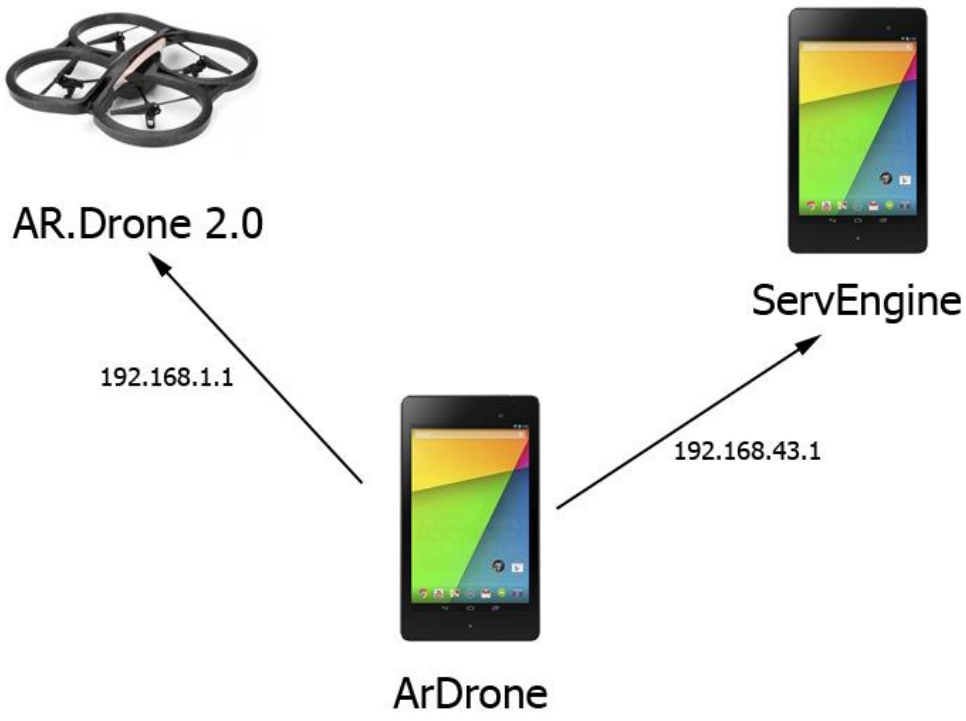


ARDrone

Esta clase contenida en el paquete "com.codeminders.ardrone" es la encargada de todo lo relacionado con el dron y por ello ha estado presente durante todo el proceso de desarrollo. A continuación se nombrarán algunas de sus características más importantes:

- Al llamar al método DroneStarter de la clase Base se crea un objeto de tipo drone. El objeto creado es una instancia de esta clase por lo que todas las llamadas que se realizan a través de dicha instancia forman parte de esta clase.
- Se determinan los puertos de envío para todos y cada uno de los tipos de comando del dron.
- Hace llamadas a las clases encargadas de generar y gestiona el hilo encargado de mandar los paquetes formateados para que el dron los pueda procesar.
- Crea y gestiona los hilos para los decoders de NavData y video.
- Cuenta con una serie de estados que indican qué tipo de acciones puede llevar a cabo el dron en función de en cuál se encuentre.

Conexiones entre aplicaciones



Pruebas

A lo largo del desarrollo de este proyecto se han realizado un gran número de pruebas las cuales podríamos dividir en tres bloques.

Pruebas de protocolo

Fueron las primeras que se realizaron y consistían en estudiar el tráfico en la red que creaba el dron para obtener datos sobre el protocolo de comunicación que era utilizado entre la aplicación oficial y el dron. Tras cada una de estas pruebas se abría lo obtenido en Wireshark y se buscaban variaciones entre unas conexiones y otras que indicasen cambios significativos como por ejemplo: un cambio en la configuración básica del dron, el estado de la batería a través de NavData, etc.

El resultado de estas pruebas fue determinante a la hora de crear la primera versión de la aplicación simuladora así como para determinar qué era imprescindible para la creación y mantenimiento de conexiones con el dron. Estos resultados son los que se comentan en el aparatado "Conociendo el protocolo" y el anexo de Wireshark.

Pruebas de funcionalidad básica

Estas pruebas consistían en conexiones con el dron y el simulador, y la comprobación de los datos de envío. Se realizaron tanto salidas por pantalla como esnifando de la red. En el caso del dron consistían en despegar y aterrizar para comprobar que todo era estable y la conexión se mantenía.

Estas pruebas permitieron comprobar el funcionamiento de las primeras funcionalidades que se añadieron. También sirvieron para comparar trazas en Wireshark de las pruebas de protocolo con éstas para comprobar que se respetaba.

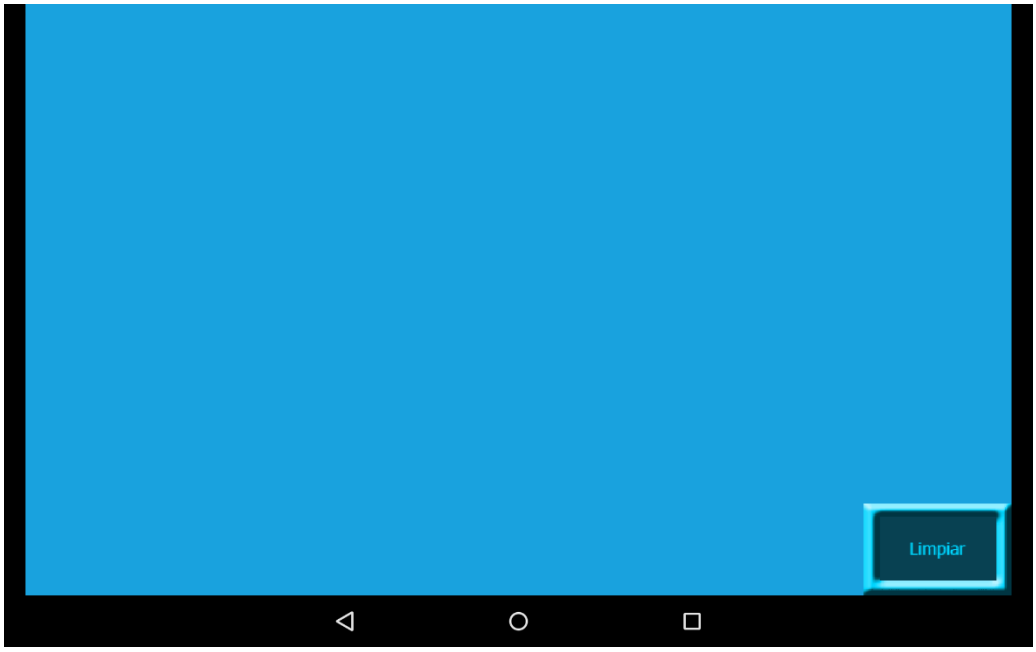
Pruebas de simulador

Son las últimas pruebas realizadas y se llevaron a cabo durante los períodos en los que no se disponía del dron físico, permitiendo que el desarrollo de la aplicación continuase.

Mediante una de estas pruebas se pudo detectar, por ejemplo, que si la aplicación se iniciaba con el dispositivo inclinado se tomaba esa inclinación como referencia. Todos los demás movimientos eran relativos a esa posición original provocando que tener el móvil en posición neutra se tomase como un movimiento en uno de los ejes. Este inconveniente de haber sido localizado durante un vuelo de prueba hubiese supuesto un problema y, probablemente, un accidente con el dron.

Las pruebas se realizaban de la siguiente manera:

1. Se activa la aplicación ServEngine

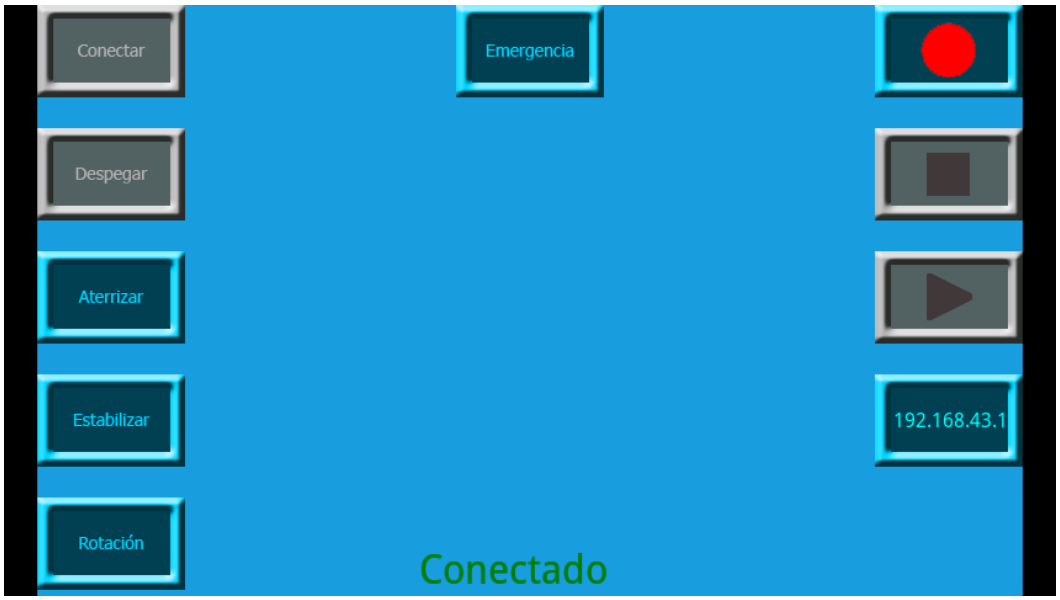


2. Con la aplicación servidor preparada procedemos a ejecutar la aplicación controladora.

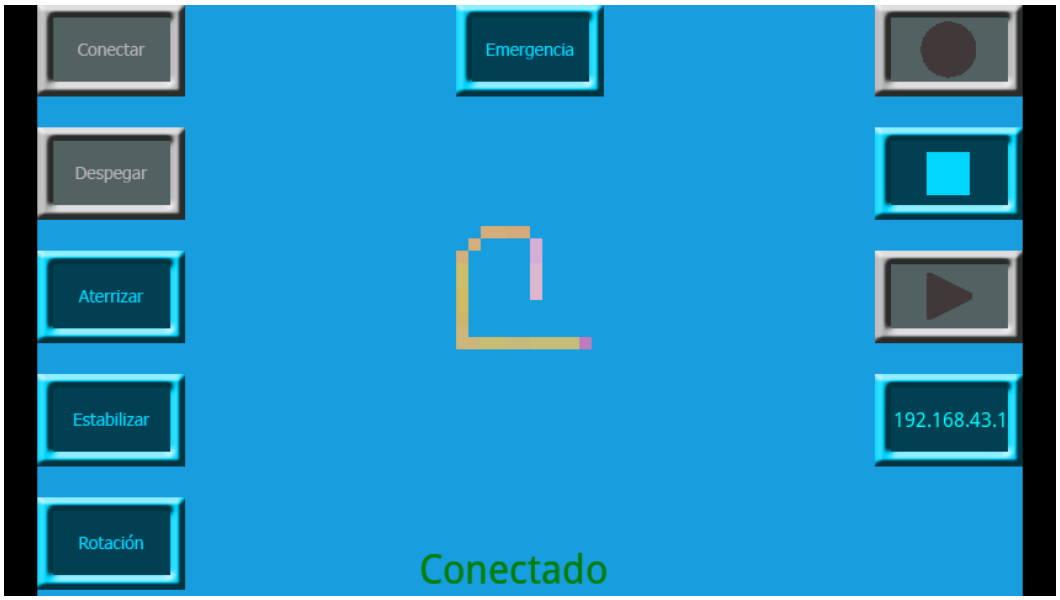
3. Nos conectamos al servidor, en este caso se encuentra en la IP 192.168.43.1

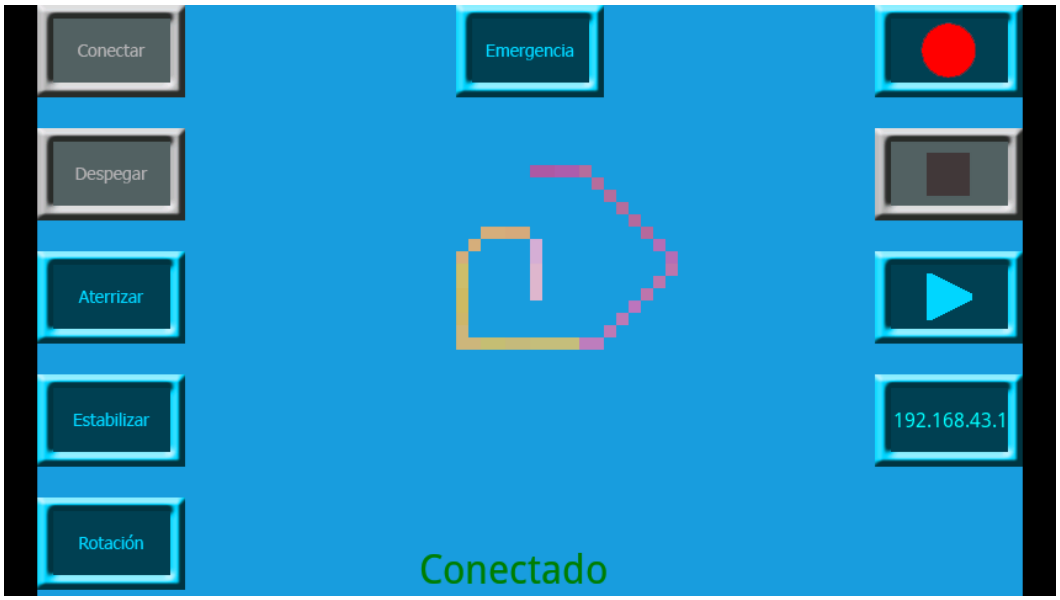
4. Una vez conectados, procedemos a despegar.

5. Cuando se ha establecido la conexión y hemos despegado ya se puede probar el control directo .

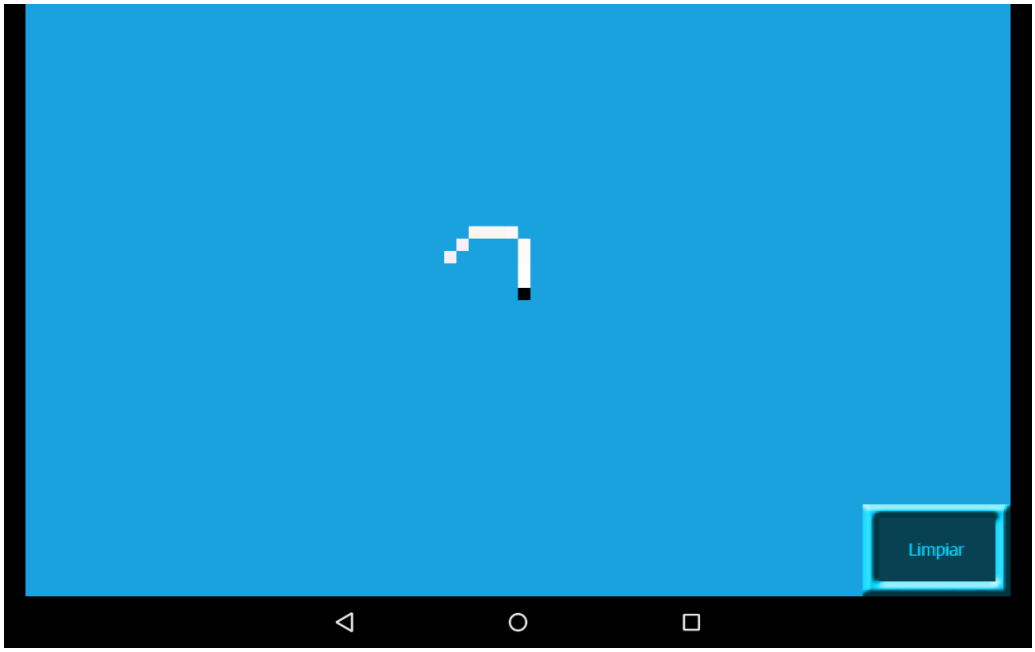


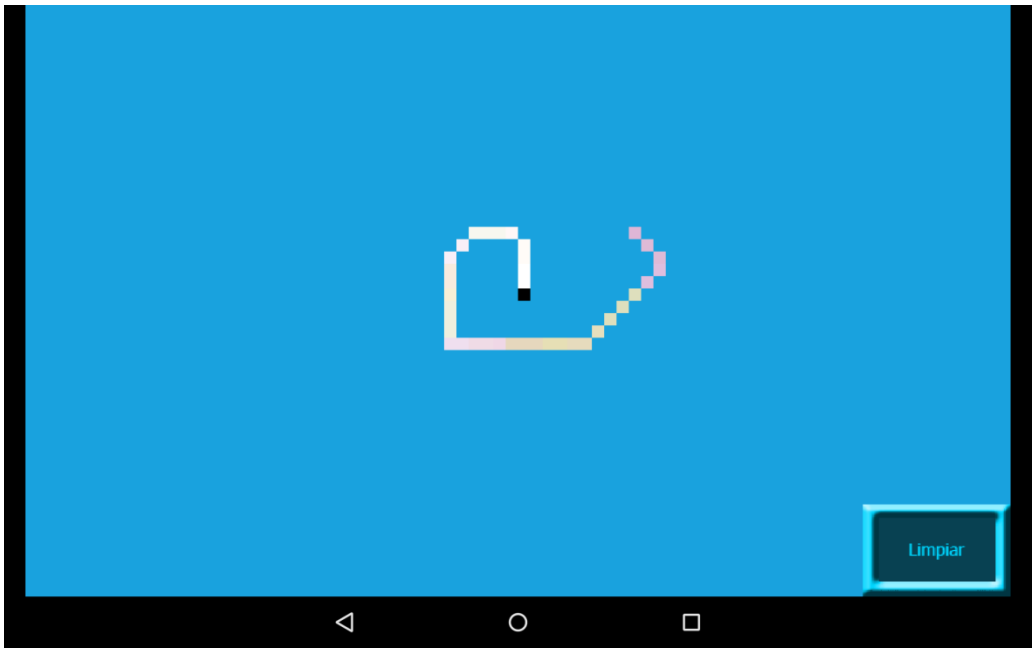
6. El control en diferido puede ser grabado en cualquier momento, pero para que el dron/simulador reciba los datos debemos estar conectados.





7. Con el recorrido grabado procedemos a su envío y observamos el resultado en el servidor.




Limpiar

Conclusiones y líneas futuras

Conclusiones

Haber tenido la posibilidad de estudiar con una aplicación comercial, de la que no soy autor, ha sido toda una experiencia ya que me ha permitido ver como se trabaja en estas empresas.

También ha sido muy interesante ver cómo funcionaba la comunidad de desarrolladores que giraba en torno al API de Parrot, aunque cabe reconocer que estaba muy abandonada a estas alturas.

El motor gráfico AndEngine, utilizado para crear los interfaces, ha sido probablemente lo mejor del proyecto ya que era algo totalmente nuevo para mí y nunca había trabajado con algo similar, aunque sí que es cierto que la ausencia de documentación hace que el proceso de aprendizaje sea más lento de lo deseado. Sin embargo, la comunidad es muy activa y tras hacer varias búsquedas se suele encontrar lo que se necesita o aproximaciones. Tampoco olvidemos que gracias a que se puede hacer referencia directa a la librería, se nos permite modificar las clases a nuestra voluntad sin necesidad de nuevas compilaciones.

La herramienta Maven es algo a tener en cuenta para proyectos de gran envergadura. Puede ser una herramienta que facilite mucho el trabajo, ya sea por su capacidad de buscar en distintos repositorios todo lo que se necesita o, al igual que AndEngine, permite modificar clases sin necesidad de tener que volver a compilar el código para generar un .jar.

La dificultad de trabajar con interfaces y controles naturales que ya había experimentado en el proyecto anterior ha vuelto a ser protagonista, en este caso, no se encontró ningún gesto natural para el control de altura que no resultase demasiado aparatoso.

Este proyecto ha tenido mucho más de investigación que de desarrollo. Salvo Eclipse todas las demás tecnologías, bien sean hardware o software, eran nuevas para mí.

En conclusión, se puede afirmar que se ha desarrollado una aplicación capaz de controlar un dron haciendo uso de todas las tecnologías descritas anteriormente.

Líneas futuras

- La implementación del eje Z. Esto vendría acompañado de un cambio en el motor gráfico para acomodar este tercer eje.
- Establecer un mecanismo que permita el envío de valores intermedios de inclinación al dron para tener un control más preciso del mismo.

- Una revisión completa y mejora de los interfaces ya que los actuales se han centrado principalmente en la funcionalidad.
- Mejorar el funcionamiento de la cámara del interfaz permitiendo que el área para dibujar los recorridos que se están grabando pudiesen abarcar más espacio.
- Implementación de un control mediante joysticks virtuales que nos proporciona AndEngine.

Bibliografía

Bibliografía digital

Eclipse

https://www.eclipse.org/

AR.Drone 2.0

http://ardrone2.parrot.com/
https://projects.ardrone.org/

Javadrone

https://code.google.com/p/javadrone/

AndEngine

http://www.andengine.org/
https://github.com/nicolasgramlich

Sensores Android

http://developer.android.com/guide/topics/sensors/sensors_motion.html

Maven

https://maven.apache.org/

Wirehsark

https://www.wireshark.org/

Otros

https://elpinguinotolkiano.wordpress.com/2014/10/27/calculo-diferencial-6-una-solucion-numerica-para-el-paracaidista/

Anexo A

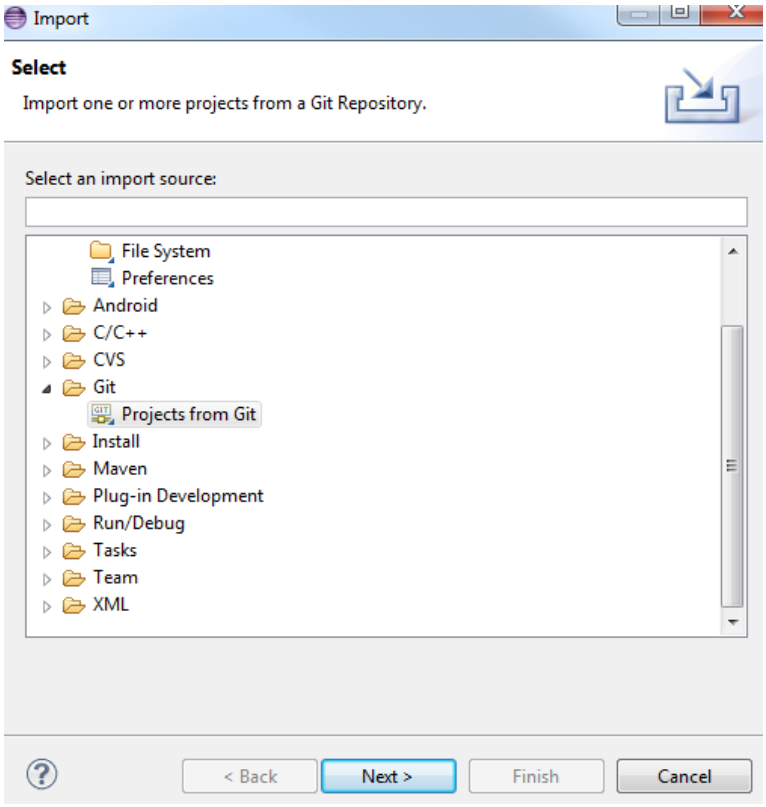
Instalación de AndEngine

Antes de comenzar a hacer uso de este motor debemos obtener la librería desde el GitHub del autor. Para ello instalamos el módulo de Git en Eclipse.

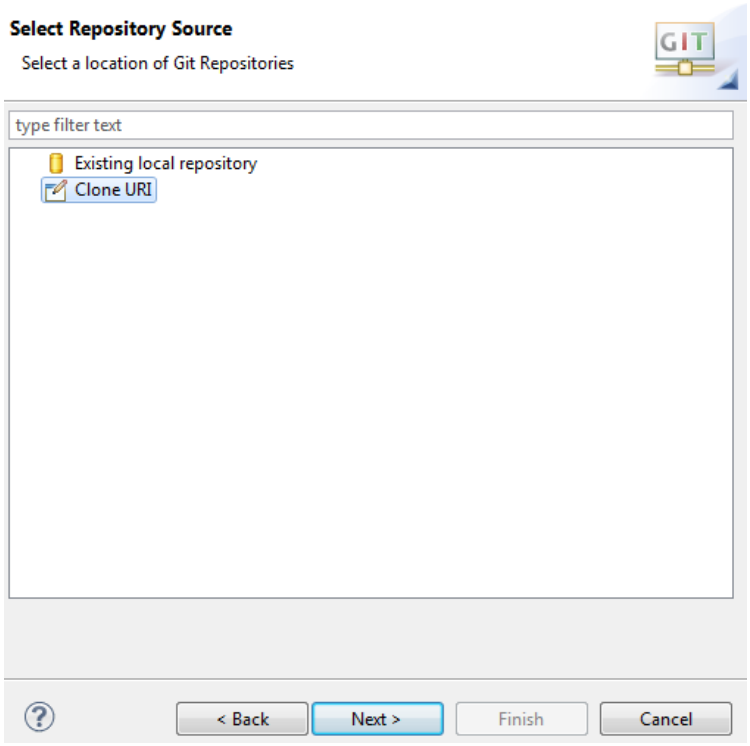
Vamos a "Install new software" y ponemos la siguiente URL:

http://download.eclipse.org/egit/updates

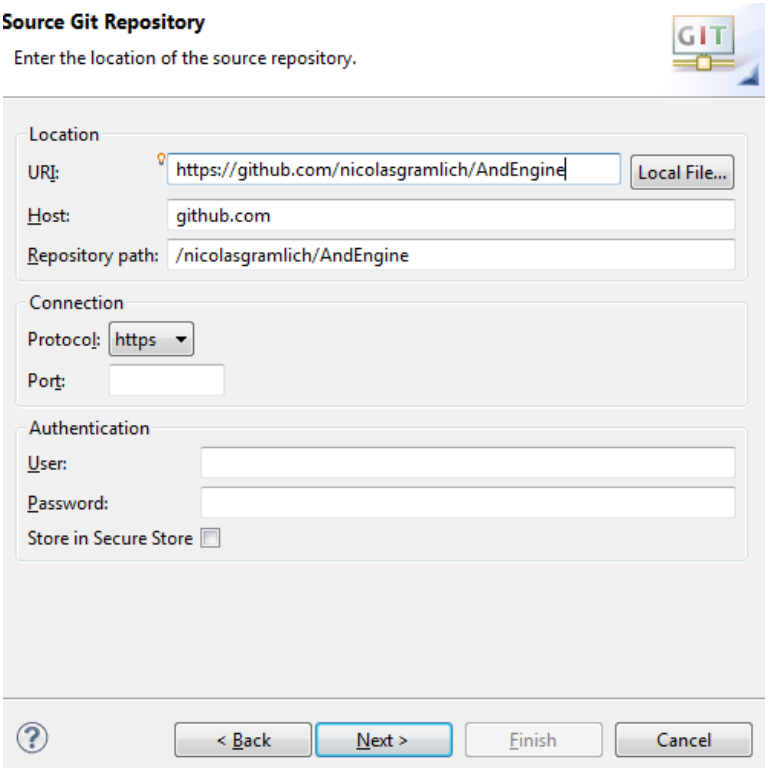
Descargamos e instalamos "Eclipse Git Team Provider → Eclipse Egit" y al seleccionar "Importar proyecto" tendremos la opción de importar desde Git



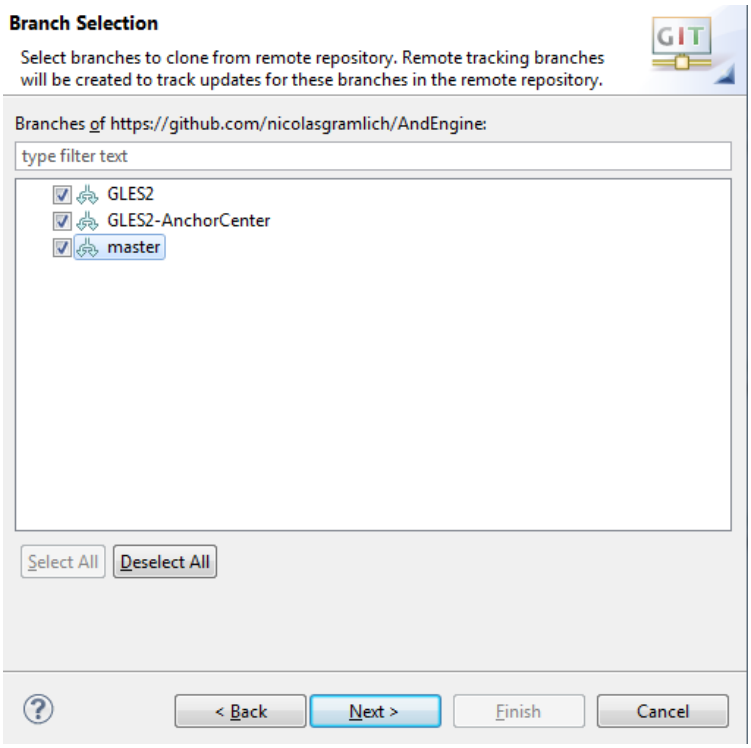
Seleccionamos clonar URI



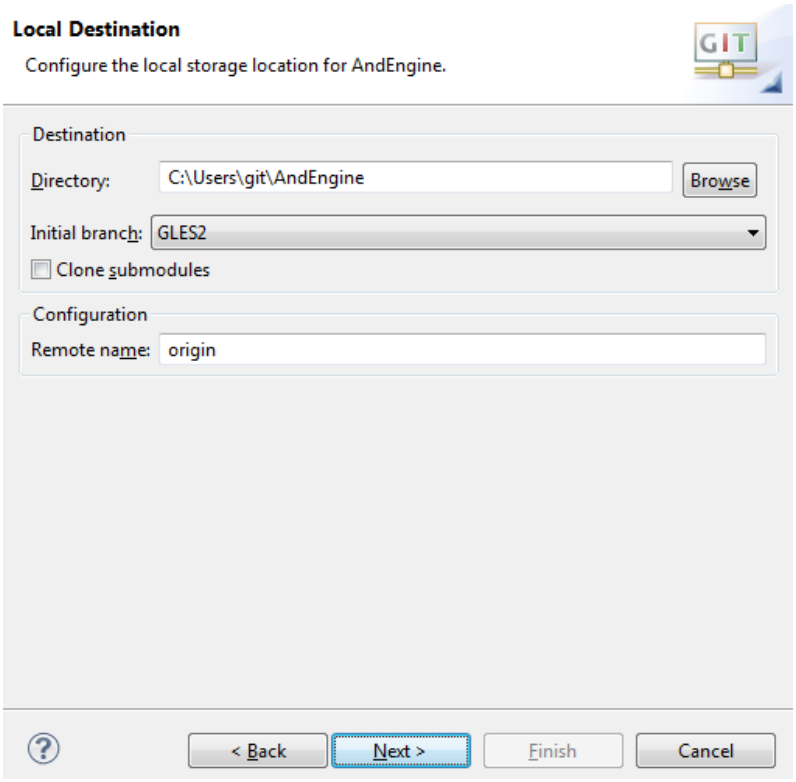
Rellenamos el campo URI con https://github.com/nicolasgramlich/AndEngine



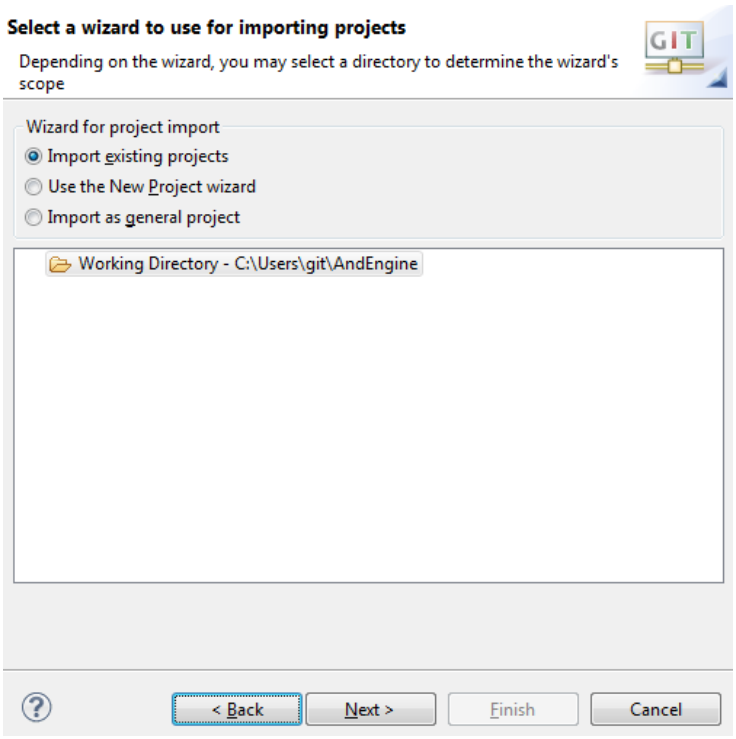
Elegimos qué queremos descargarnos.



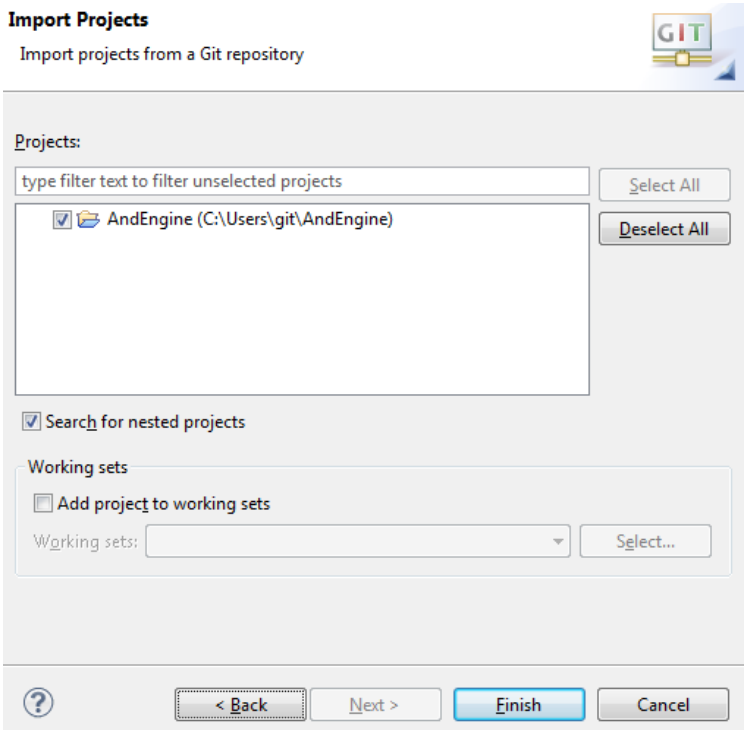
Y definimos el destino.



Cuando se ha finalizado la descarga se solicitará qué hacer con el proyecto o proyectos descargados.

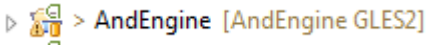


Elegimos importar los proyectos existentes.

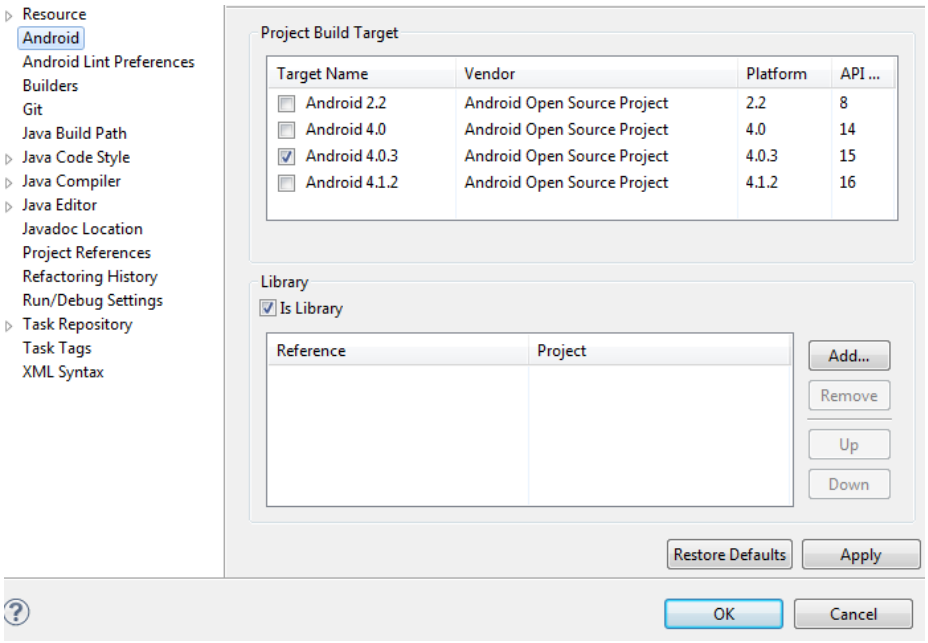


Y finalmente especificamos qué queremos importar.

Una vez se han realizado los pasos anteriores el proyecto AndEngine aparecerá en nuestro espacio de trabajo



Con el proyecto ya en nuestro espacio de trabajo sólo falta marcar que se trata de una librería. Para ellos vamos a Propiedades y en Android nos aseguramos de que el checkbox "Is library" está marcado.

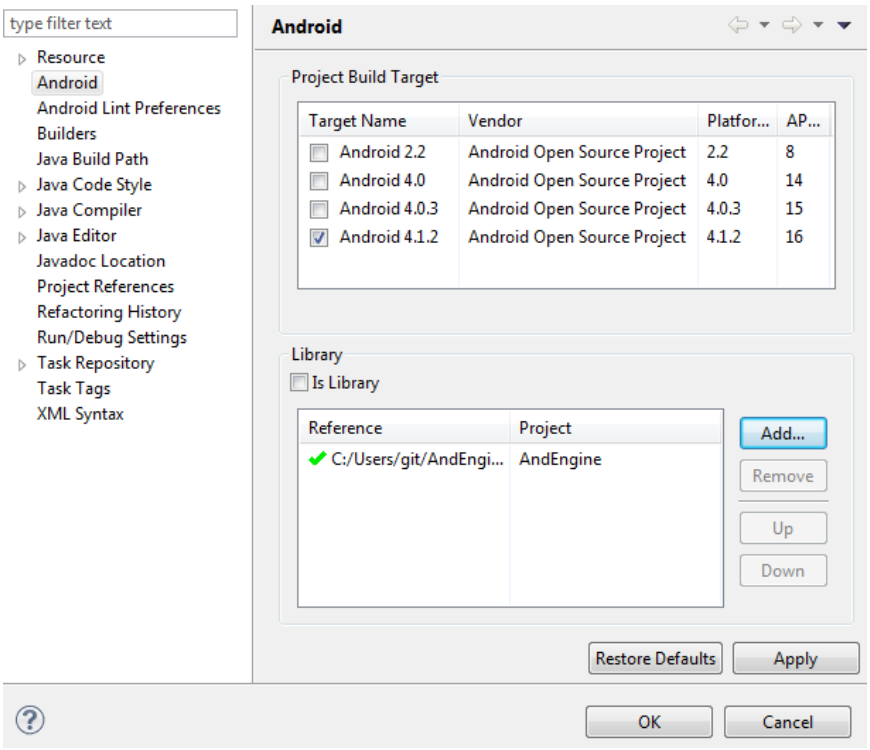


Elegimos la versión de Android que queremos que compile AndEngine y ya está lista para usarse.

Añadiendo AndEngine a un proyecto

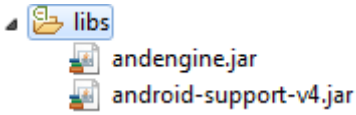
Podemos usar AndEngine de dos maneras:

1. La más recomendable consiste en añadir a un proyecto la referencia a AndEngine. Para hacer esto nos basta con ir a "Propiedades del proyecto → Android" y añadirlo.



Una vez añadido ya podemos hacer uso del motor gráfico.

2. Otra opción menos recomendable, ya que no nos permitiría modificar lo que necesitemos del motor gráfico sin recompilarlo, es copiar el archivo .jar que genera el proyecto de AndEngine en la carpeta bin a la carpeta libs del proyecto que va a hacer uso del motor.



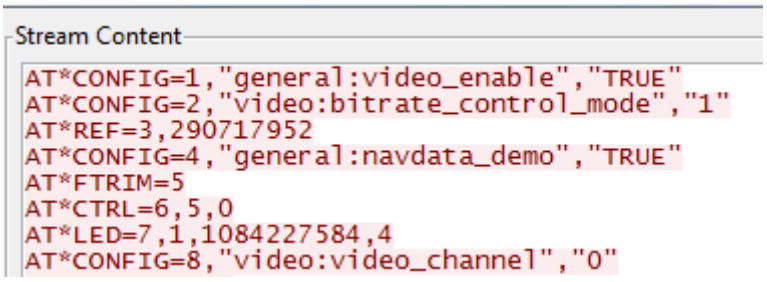
Anexo B

Utilización de Wireshark

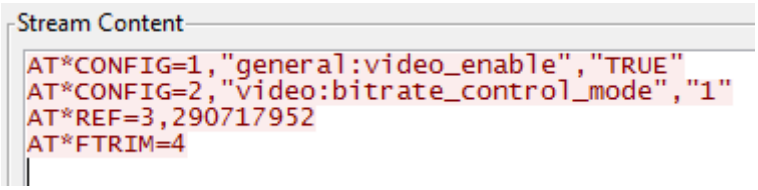
El programa Wireshark ha sido utilizado constantemente durante el proyecto para determinar el estado de las conexiones y cómo se establecían.

Todas las pruebas de Wireshark consistían en guardar el tráfico de la red y realizar conexiones, desconexiones y pruebas de vuelo. Una vez se tenían los datos se procedía a simular las conexiones.

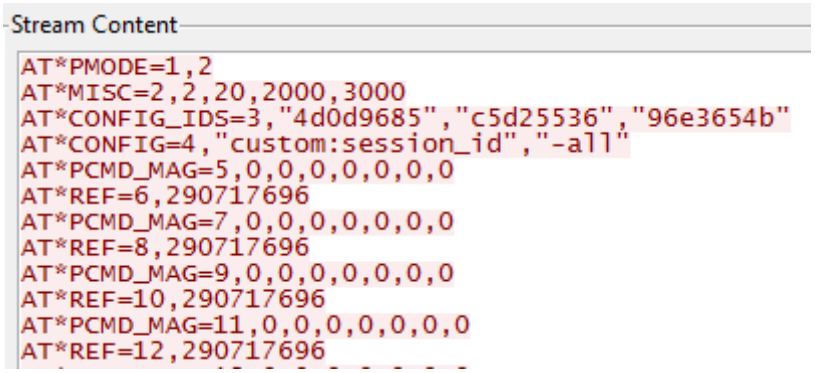
El siguiente ejemplo se obtuvo durante una prueba con el código original de Codeminders:



Y el siguiente en una conexión simulada con nuestra aplicación servidor:



Como se puede observar siguen unas pautas similares, aunque cabe destacar que se redujo el envío de información, ya que había paquetes que no eran determinantes para el control básico del dron. Esto resulta más que evidente si comparamos una traza de la aplicación oficial.



Anexo C

Manual de Usuario

ARDrone

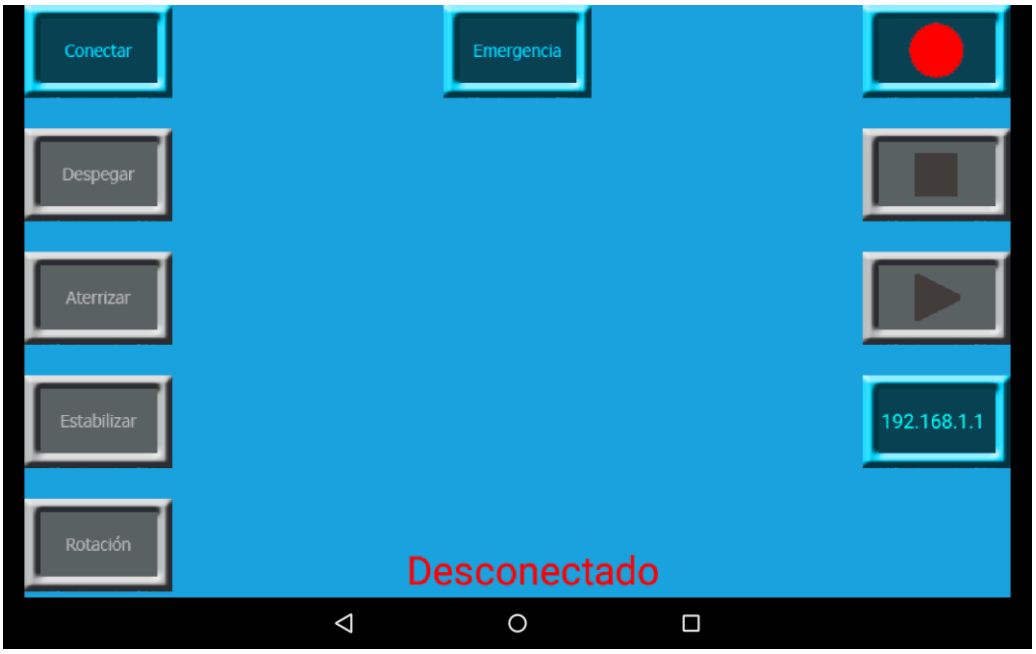
Consideraciones sobre la aplicación

Esta aplicación se basa en el protocolo desarrollado por Codeminders en su proyecto JavaDrone. Antes de hacer uso de la misma conviene comprobar que el dispositivo es compatible con la implementación original.

Es necesario abrir la aplicación con el móvil en una posición paralela al suelo o sobre una superficie horizontal más elevada ya que los sensores se calibran de manera automática al abrir la aplicación.

El interfaz

Al abrir la aplicación aparecerá el interfaz único desde el que podemos controlar el funcionamiento del dron.



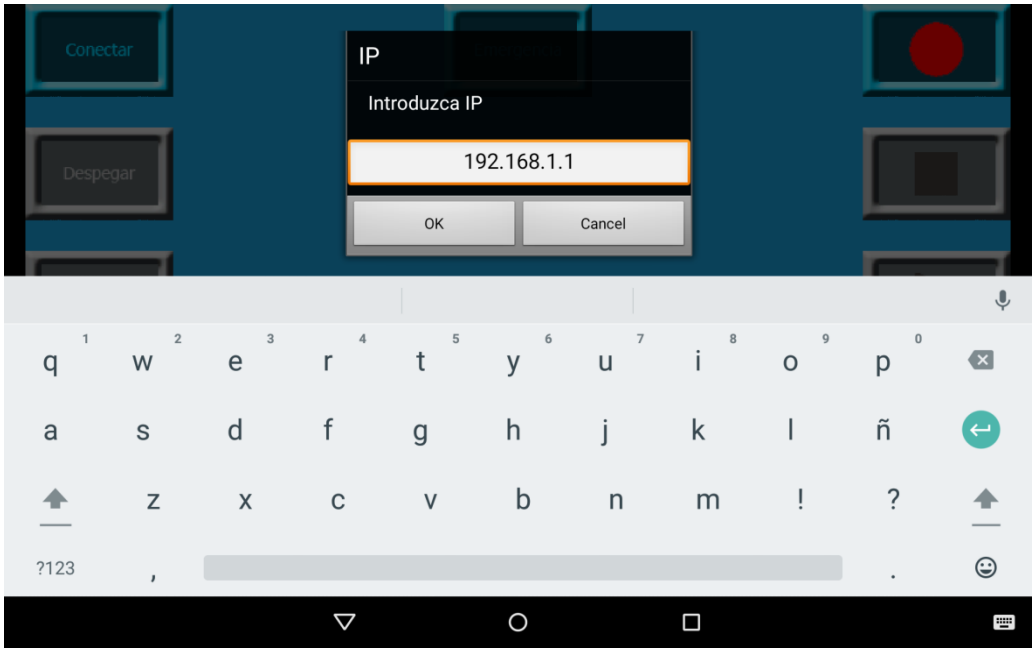
Botones en la aplicación

- **IP**: Cuarto botón del lado derecho. En él se observa la IP a la que nos disponemos a conectarnos.
- **Conectar**: Una vez definida la IP a la que deseamos conectarnos se encarga de realizar la conexión al dispositivo. Ya sea un dron o un simulador.
- **Despegar**: Hace despegar al dron y permite el envío de instrucciones a éste.
- **Aterrizar**: Ordena al dron realizar un aterrizaje sobre su posición actual.

- **Estabilizar**: Ordena al dron estabilizarse en la posición en la que esté.
- **Rotación**: Cambia el funcionamiento de las inclinaciones laterales del dispositivo. En vez de que la inclinación provoque movimientos laterales en el dron, le permite rotar a ese lado manteniendo la posición.
- **Grabar**: Una vez pulsado comienza a almacenar y dibujar en la pantalla el recorrido que estamos generando.
- **Parar**: Detiene el proceso de grabación.
- **Reproducir**: Envía al dron o al simulador el recorrido guardado para su ejecución.
- **Emergencia**: Solicita una parada de emergencia, intenta aterrizar y finalmente para los motores en vuelo.

Conectándonos al dron

El texto inferior nos indica el estado de la conexión. Para poder conectarnos es necesario activar el WiFi del dispositivo. Tras comprobar que el WiFi está conectado debemos elegir la IP con la que queremos establecer la comunicación. Por defecto está la dirección IP típica de un AR.Drone 2.0 de Parrot. Para definir la IP basta con pulsar el botón en el que aparece y un diálogo nos permitirá cambiarla.



Una vez definida la IP a la que deseamos conectarnos pulsamos conectar. Si todo ha ido bien el interfaz será el siguiente:

El dron en vuelo

Una vez se pulsa despegar se activa el modo vuelo para la aplicación. Desde ese momento, los movimientos que afecten a la inclinación del dispositivo son convertidos en instrucciones para el dron o simulador. Una inclinación a la izquierda provoca un movimiento a ese mismo lado por parte del dron y a la derecha lo mismo, pero a la inversa.

Por otro lado, una inclinación del dispositivo hacia delante ordena avanzar al dron y la inclinación hacia atrás ordena lo contrario.

Además estarán activas las opciones aterrizar, estabilizar y rotación.

El botón aterrizar ordenará al dron a aterrizar y devolverá la aplicación al estado de pre-conexión.

El botón estabilizar provocará que el dron se estabilice de forma paralela a la superficie sobre la que se encuentre.

El botón rotación transforma el movimiento lateral a izquierda y derecha por rotación en estas mismas direcciones con el dron estabilizado.

Grabando

Esta opción estará disponible incluso si la conexión no está activa, ya que podemos querer preparar un recorrido de antemano cuando el dron haya despegado. Sin embargo, para que las órdenes sean efectivas debemos asegurarnos de que estamos conectados bien al dron o al simulador o se perderán.

Con el botón de grabar pulsado comenzamos a mostrar por pantalla el recorrido que se está grabando.

Tras pulsar detener podemos observar el recorrido realizado y se nos permite sobrescribirlo, volviendo a grabar, o mandarlo al dispositivo que esté escuchando al otro lado.

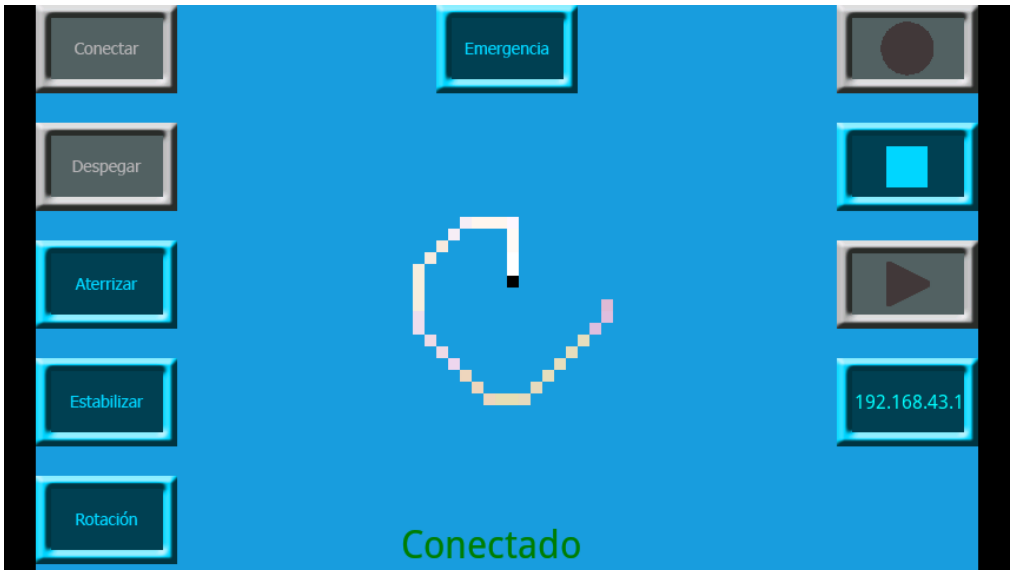
Durante el envío se inhabilitan los botones de grabar y reproducir para evitar contradicciones, aunque se sigue permitiendo el envío de instrucciones desde el dispositivo y se mantienen las funcionalidades de los botones de emergencia y del lado izquierdo por si hubiese algún problema con el recorrido grabado.

Al volver a poner a grabar al dispositivo, el interfaz se limpia automáticamente.

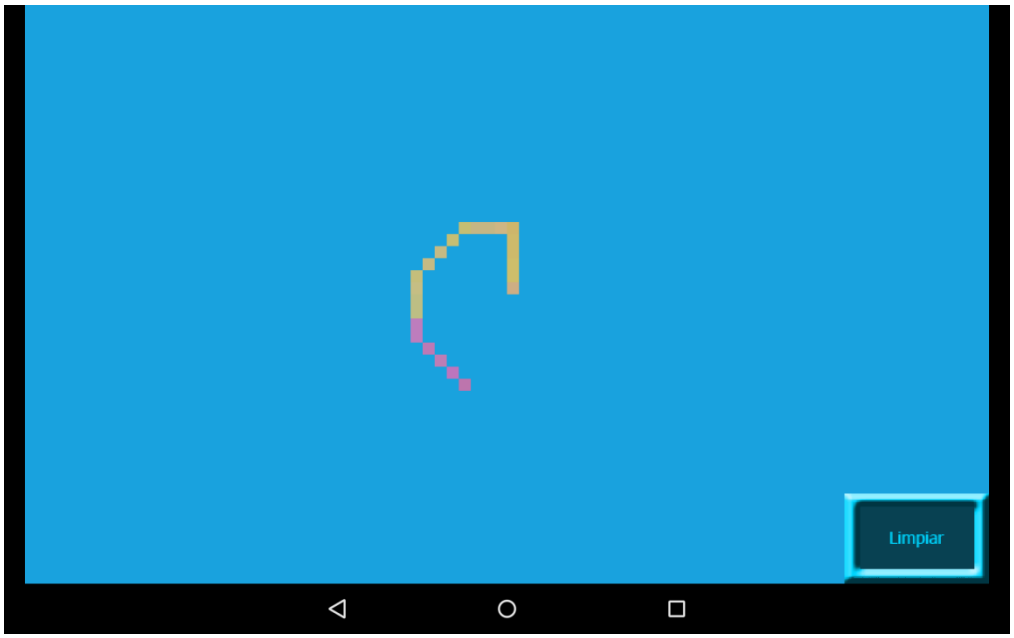
ServEngine

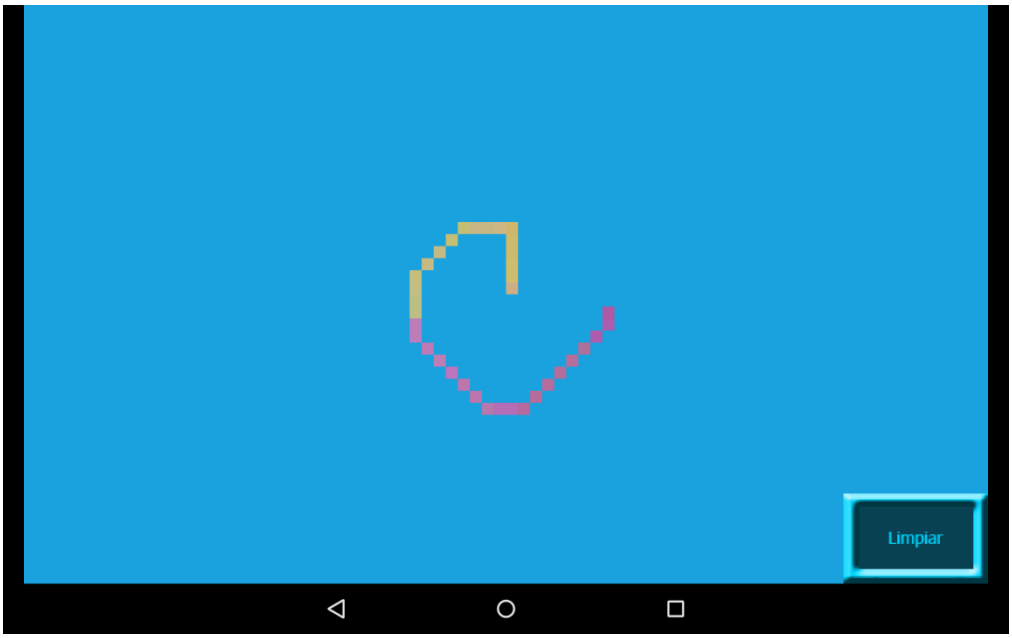
El funcionamiento de esta aplicación es muy simple. Una aplicación de control conectada a este simulador nos permitirá ver qué tipo de instrucciones se estarían mandando al dron. Cuenta con un botón que nos da la posibilidad de resetear el interfaz para distinguir mejor los recorridos planteados.

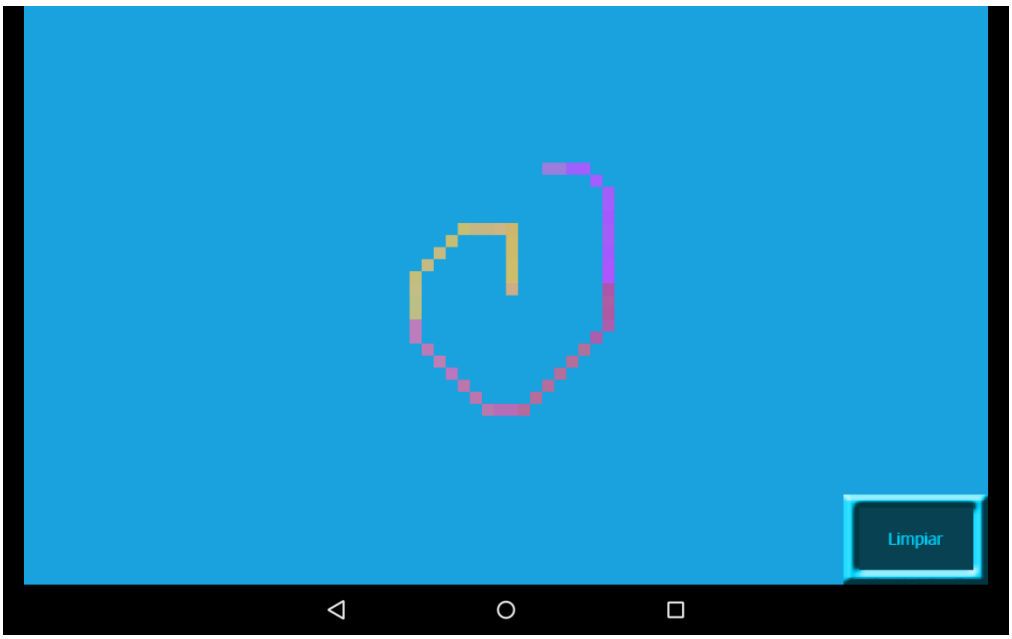
A continuación un ejemplo de envío en diferido. La primera imagen es enviada desde la aplicación ARDrone.



La aplicación servidor recibe la información de este recorrido y lo muestra por pantalla en varias fases:







Con el envío ya completado podemos proceder a limpiar la pantalla a la espera de las próximas instrucciones.

Desarrollo de aplicaciones móviles para AR.Drone Parrot en AndEngine

Autor: Jorge Wederago Jiménez

Tutor: Óscar Ardaiz Villanueva

Pamplona, 29 de Junio 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Contenido de la presentación

- Introducción
- Objeto
- AR.Drone 2.0
- AndEngine
- Desarrollo de la aplicación
- Pruebas
- Conclusiones
- Líneas futuras

Introducción

- Aplicación de tecnología a toda clase de campos: militares, salvamento, exploración, deportes, ocio…
- La gran demanda provoca reducción en sus costes

Objeto

- Aplicación para controlar un dron
- Desarrollado en Java/Android
- Control intuitivo
- Interfaz gráfico en AndEngine

AR.Drone 2.0

- Registrador de vuelo GPS
- Cámara HD a 720p a 30 FPS
- Peso total de 380g con la carcasa de exteriores y 420 con la de interiores.
- Motores “inrunner” sin escobillas 14,5W 28.500 RPM
- Procesador de 1 GHz y 32 bit ARM Cortex A8 con vídeo DSP TMS320DMC64x de 800 MHz
- Linux 2.6.32
- RAM DDR2 de 1GB a 200MHz
- USB 2.0 de alta velocidad para extensione Wi-Fi b g n
- Giroscopio de 3 ejes con una precisión de 2000°/seg.
- Acelerómetro de 3 ejes con una precisión de +/- 50mg
- Magnetómetro de 3 ejes con una precisión de 6°
- Sensor de presión con una precisión de +/- 10 Pa
- Sensores de ultrasonido para medir la altitud de avance
- Cámara vertical QVGA de 60 FPS para medir la velocidad de avance

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AR.Drone 2.0

- FreeFlight

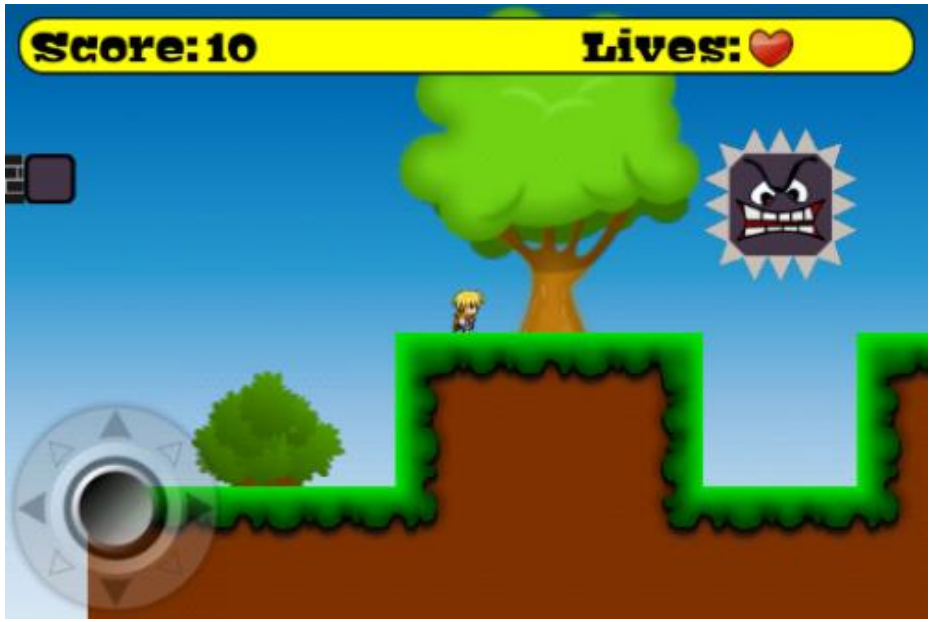
AR.Drone 2.0

- FreeFlight

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AndEngine

- Desarrollado por Nicolas Gramlich
- OpenGL ES
- Java/Android
- 2D
- Escalado Automático
- Sin documentación
- Gran comunidad



Desarrollo de la aplicación

- Protocolo
  - Comandos AT*___
    - REF
    - PCMD
    - LED
    - ….
  - Puertos
    - 5551: FTP
    - 5554: NavData
    - 5555: Video
    - 5556: Comandos
    - 5559: Control

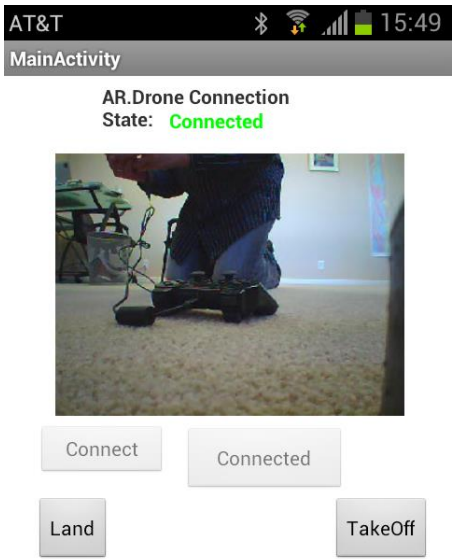
```
.......H 6.AT*CTR
L=22,5,0 .AT*PCMD
_MAG=23, 0,0,0,0,
0,0,0.AT *REF=24,
29071769 6.
```

Desarrollo de la aplicación

- Simulando el dron
  - Mission Planner y FlightGear
    - Necesidad de una placa Arduino/Raspberry
  - Implementación propia en Android
    - Clase MainActivity.java
    - Clase HiloServidor.java

Desarrollo de la aplicación

- Imposibilidad de elegir a dónde conectarnos
  - JavaDrone de Codeminders
    - Utilización de Maven
    - Interfaz sencillo
    - Controles con mando Dual Shock 3



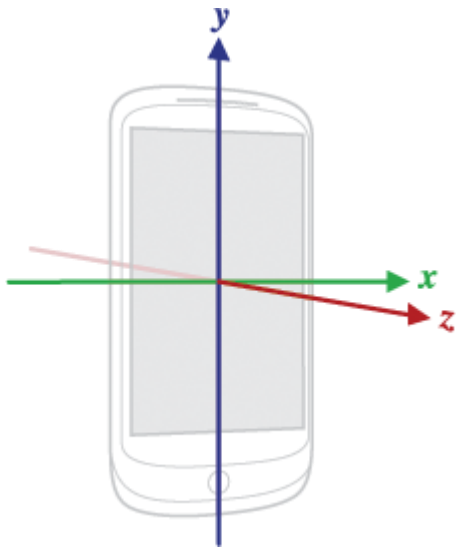
Desarrollo de la aplicación

- Creación aplicación controladora
  - Entender el funcionamiento de JavaDrone
  - Clase DroneStarter
    - Modificación de la DEFAULT_DRONE_IP
    - Añadir a connect()

```
changeState(State.CONNECTING);
changeState(State.DEMO);
```

Desarrollo de la aplicación

- Implementación para control de vuelo
  - Uso de sensores de dispositivos Android
    - SensorEventListener
      - onAccuracyChanged
      - onSensorChanged
    - Tratamiento de las lecturas
    - Llamadas a drone.move()



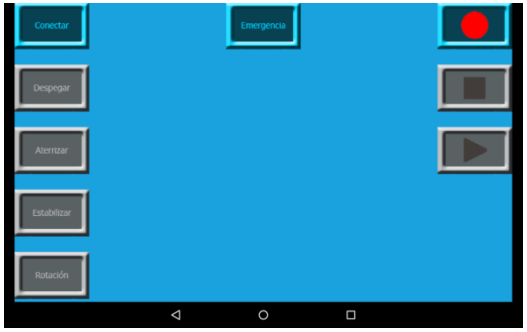
Desarrollo de la aplicación

- Otras ideas
  - Calcular distancias recorridas según el valor del acelerómetro
  - Grabación y retransmisión de órdenes

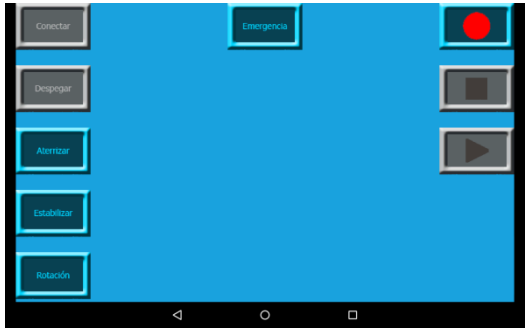
Desarrollo de la aplicación

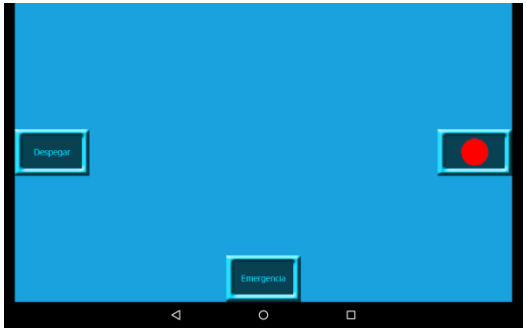
- Interfaz
  - AndEngine
    - Implementar SimpleBaseGameActivity
      - onCreateEngineOptions()
      - onCreateResources()
      - onCreateScene()

Desarrollo de la aplicación

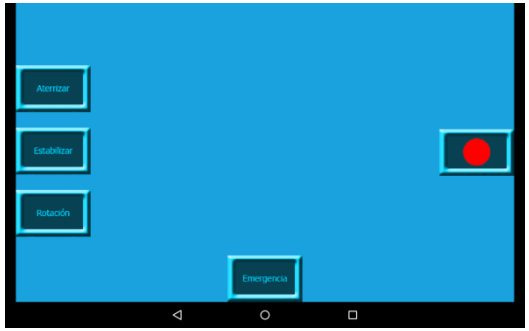


→





→

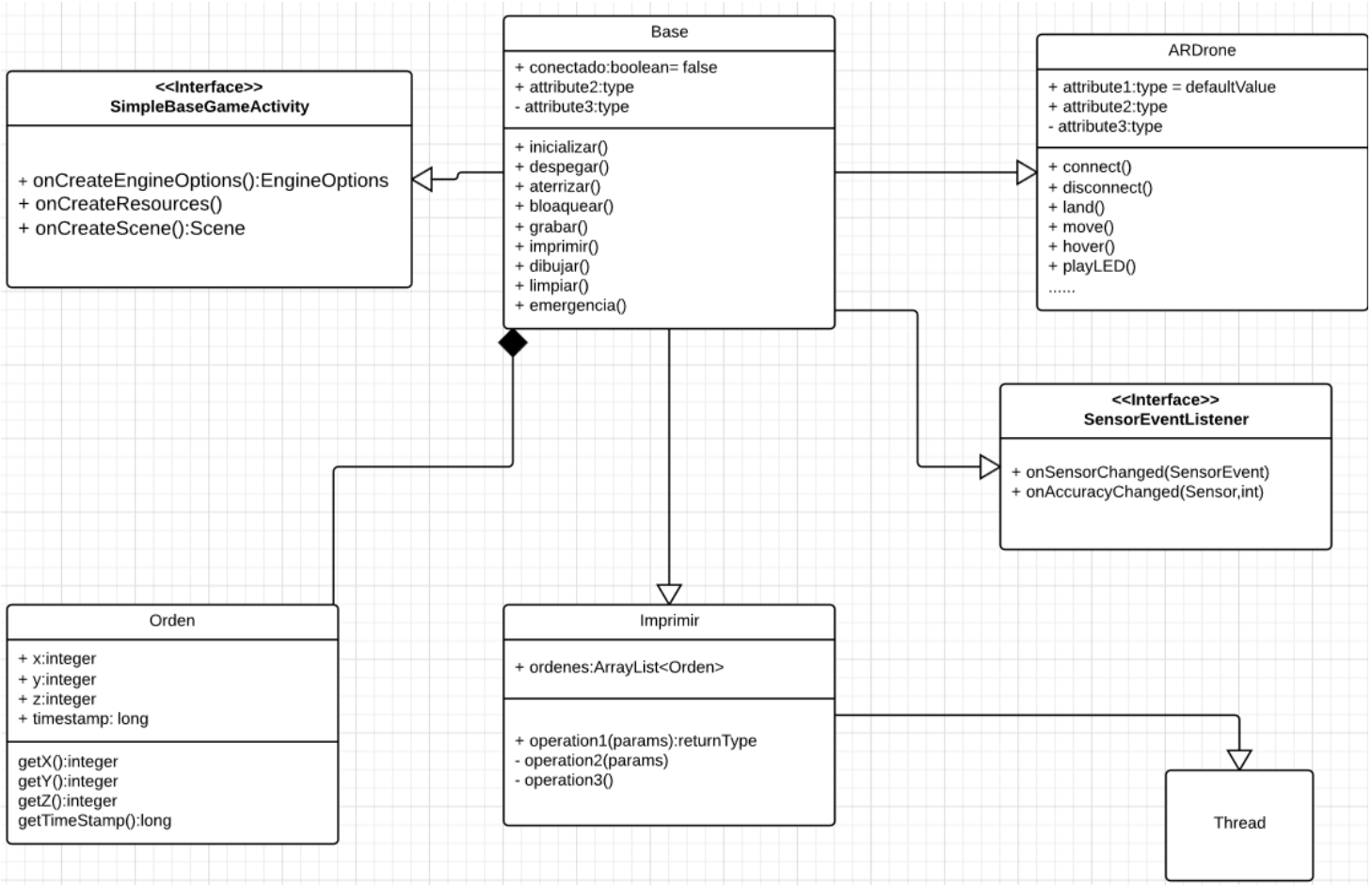


Desarrollo de la aplicación

- ServEngine
  - HiloServidor()
  - MainActivity()

- ArDrone
  - Orden
  - Imprimir
  - InputText
  - Base

Desarrollo de la aplicación



upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Pruebas

- Protocolo
  - Estudio del tráfico en la red
- Funcionalidad básica
  - Comprobar si el protocolo se reproduce correctamente
- Simulador
  - Pruebas realizadas sin el dron físico

Conclusiones

- Posibilidad de estudiar una aplicación comercial
- La comunidad de desarrolladores
- Primera experiencia con un motor gráfico
- Descubrimiento de Maven
- Ventajas y desventajas de movimientos naturales
- Más investigación que desarrollo
- Obtención de una aplicación controladora funcional

Líneas futuras

- Implementación total del eje Z
- Envíos más precisos desde el dron
- Revisión y mejora de los interfaces
- Mejor utilización de las cámaras en AndEngine
- Controles virtuales con AndEngine

Desarrollo de aplicaciones móviles para AR.Drone Parrot en AndEngine

Autor: Jorge Wederago Jiménez

Tutor: Óscar Ardaiz Villanueva

Pamplona, 29 de Junio 2015

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa